# Diseño y desarrollo de un sistema de geolocalización activos

## MEMORIA TRABAJO FINAL DE GRADO




*Raúl Valverde Sánchez*

GRADO EN INGENIERÍA INFORMÁTICA

TECNOLOGÍAS DE LA INFORMACIÓN

UNIVERSITAT POLITÈCNICA DE CATALUNYA - 2019

DIRECTOR: MANUEL MOLERO RUIZ

PONENTE: ENRIQUE ROMERO MERINO

EMPRESA: SERVICETONIC S.L.

FECHA DE DEFENSA: 04/07/2019

# TABLA DE CONTENIDO

# 1. INTRODUCCIÓN

## 1.1. CONTEXTO Y ACTORES

El proyecto se basa en el diseño, desarrollo e integración de una mejora dentro de un *software* de automatización y administración de servicios. Este *software* es distribuido por la empresa ServiceTonic S.L. bajo licencia.

La herramienta que desarrolla y proporciona ServiceTonic es utilizada por empresas de varios países para optimizar sus servicios e incrementar su productividad. Hay varios servicios proporcionados por este *software* como: gestión de tickets y problemas, CMDB (gestión de recursos), *SLAs*, catálogo de servicios, monitorización de red, entre otros.

Al hablar de gestión de tickets y problemas siempre pongo el mismo ejemplo a quien me pregunta cómo o qué es ServiceTonic:

*"Imagínate que estás en tu puesto de trabajo y resulta que necesitas imprimir un documento y la impresora necesita un cambio de cartucho de tinta. Entonces, tú como empleado abrirías una incidencia especificando el motivo y se asignaría a un agente el cuál sería responsable de solucionar el problema."*

El acceso a la herramienta se hace mediante el navegador web, por lo que hace de ServiceTonic una aplicación web accesible desde el navegador. El hecho de que este *software* esté construido para que sea accesible totalmente desde el navegador web hace que la herramienta sea multiplataforma (PC, *tablet* y móvil).

El aspecto que tiene la herramienta es el siguiente:

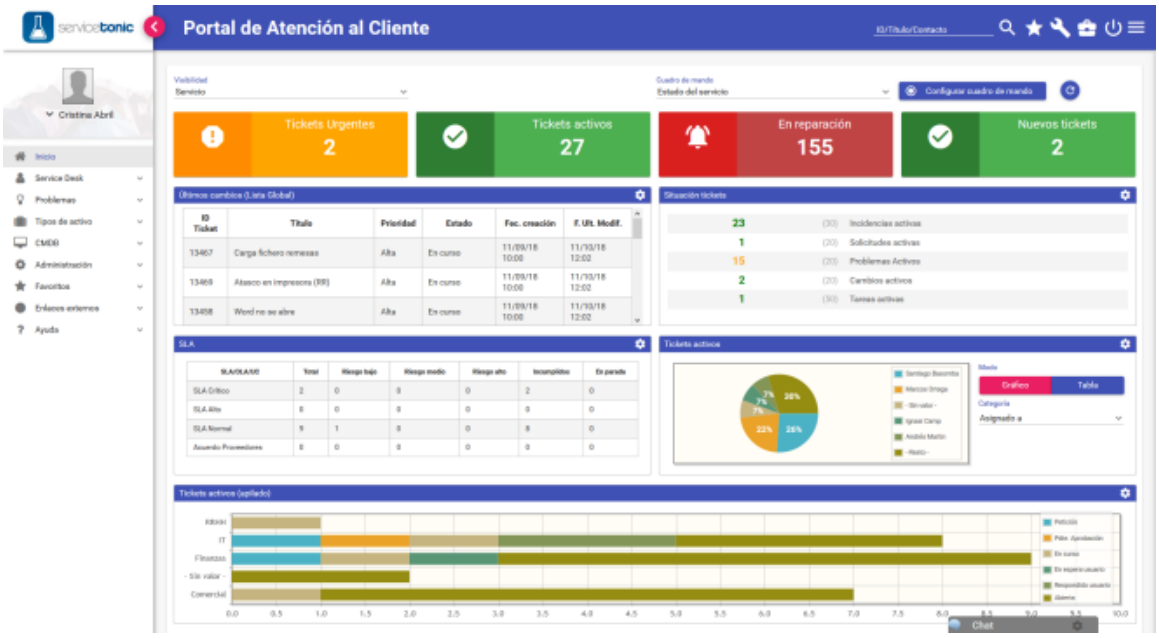


*Figura 1.1.1. Aspecto de la pantalla principal de la aplicación de ServiceTonic*

El tablero o pantalla principal muestra información en tiempo real sobre aquellos indicadores más críticos del servicio a monitorizar o administrar. Además, es totalmente configurable según las necesidades del servicio y con acceso a distintas vistas. Se pueden mostrar elementos de todo tipo como gráficas, paneles, calendarios, entre otros, para representar las incidencias del servicio a monitorizar.

La herramienta es comercializada, como he descrito antes, a aquellas empresas que quieran optimizar o monitorizar sus servicios internos bajo una licencia. El papel posterior que desempeña ServiceTonic es la de incorporación de nuevas mejoras, el mantenimiento de su producto y proporcionar un servicio de *HelpDes*k para resolver las posibles dudas o incidencias que los clientes puedan tener con la herramienta. A parte, se pueden solicitar reuniones para explicar cara a cara cómo utilizar este software.

Las empresas que son clientes de ServiceTonic S.L. juegan un papel muy importante dentro de la empresa ya que son las que demandan cuáles son las próximas mejoras que les gustaría ver en las próximas versiones. Dicho esto, una de las mejoras más solicitadas por los clientes fue, entre otras, la posibilidad de geolocalizar los diferentes activos administrables con la herramienta dentro de su empresa. Cuando hablamos de los diferentes activos, los clientes se refieren a:

- La posibilidad de conocer la geolocalización donde se ha producido una incidencia (un ticket).
- Conocer la geolocalización de sus agentes y contactos.
- Geolocalizar sus recursos físicos como impresoras, *tablets*, *PCs*, etc.

Así pues, el producto está dirigido en su totalidad, como he mencionado antes, a los clientes que deseen utilizarlo bajo una licencia previamente adquirida. Estos clientes son empresas de cualquier tamaño o número de empleados ya que la herramienta se adapta totalmente a las necesidades de la empresa que lo utilice.

Esta funcionalidad supondrá un suplemento a las actuales licencias distribuidas por ServiceTonic a sus clientes ya que el servicio de geolocalización es ajeno a nosotros debido a que utilizaremos los servicios de Google como comentaremos más adelante.

## 1.2. ESTADO DEL ARTE

El estado del arte de este proyecto se centra en estudiar qué posibilidades existen dentro de los diferentes servicios de geolocalización.

Es necesario remarcar que en el momento que se barajaron las diferentes posibilidades de los distintos servicios de geolocalización se descartó en un primer momento el uso de la API de *Google Maps* ya que por aquel momento el precio de la tarifa era fijo y bastante elevado. Posteriormente se acabaría escogiendo debido al cambio de tarifas incluyendo el pago por uso y evidentemente a la potente infraestructura que posee Google entorno a sus servicios de geolocalización.

Por otro lado, existen varias herramientas similares a ServiceTonic que utilizan tecnologías de geolocalización, ya sea de empleados, agentes, contactos o cualquier activo de la empresa. Por lo que se ha podido averiguar sobre dichas herramientas, todas utilizan los servicios de geolocalización ofrecidos por Google. Esto es debido a la potente infraestructura que posee Google y las posibilidades que ofrecen a través de sus servicios.

Ejemplos sobre herramientas que ofrezcan la funcionalidad propuesta en este proyecto son:

- *HelloTracks* (hellotracks.com): Ofrecen una plataforma bastante completa donde es posible visibilizar a los empleados y visualizar diferentes incidencias y eventos laborales. Por otro lado, es una herramienta que se centra básicamente en ofrecer ese servicio, no es una herramienta de gestión de servicios en sí, como lo es ServiceTonic. En su página web se definen como una herramienta de geolocalización y seguimiento.

- *JustSamIt* (justsamit.com): Es una herramienta IT de administración de recursos, todo tipo de dispositivos. Al igual que la herramienta anterior se centra en un solo tipo de activo, en este caso relacionados con todo tipo de *hardware*.

- *Freshdesk* (freshdesk.com): Posiblemente la herramienta más similar a ServiceTonic ya que ofrece servicios muy similares como gestión de incidencias a través de *tickets*, gestión de SLAs[1], etc.

Dicho esto, a día de hoy las tecnologías sobre geolocalización están bastante desarrolladas y son muy precisas, sobre todo en dispositivos móviles donde se hace uso de tres sensores (*WiFi*, *LTE* y *GPS*) para triangular la posición con mayor exactitud.

Como comentario final, deberemos paliar con un obstáculo que más adelante comentaremos sobre la geolocalización en dispositivos móviles en segundo plano.

1. SLA: es un acuerdo escrito entre un proveedor de servicio y su cliente con objeto de fijar el nivel acordado para la calidad de dicho servicio.

## 1.3. FORMULACIÓN DEL PROBLEMA

Así pues, después de reunir todas las recomendaciones y peticiones sobre cómo querían que fuera esta nueva funcionalidad se llegó a formular definitivamente lo que sería este proyecto.

El problema que hace falta resolver por parte de la empresa es el desarrollo y la integración de una mejora sobre el software de ServiceTonic donde las funcionalidades descritas anteriormente puedan ser geolocalizadas aportando un mayor control sobre estas y optimizando los procesos y servicios internos del cliente que utilice el software.

La solución pasa por adaptar una solución parcial existente, como son los servicios que nos proporciona la API de Google *Maps*, al software de ServiceTonic adecuándolo a las necesidades de los objetivos definidos. Se diseñarán las interfaces gráficas necesarias para que el usuario pueda establecer los parámetros necesarios para poder interactuar con la nueva funcionalidad adecuadamente.

Actualmente los clientes si quieren gestionar la geolocalización de sus activos, ya sean tickets, agentes o elementos hardware lo han de hacer manualmente introduciendo los datos en el anexo del elemento en cuestión. Es por eso, que se considera en la empresa una funcionalidad que podría tener una muy buena acogida agilizando todo este proceso.

## 1.4. OBJETIVOS

A continuación, se describirán los objetivos a cumplir durante la realización del proyecto de los cuales los beneficiarios serán la empresa y sus clientes, por un lado, ya que obtendrán un servicio de más calidad y más completo y ServiceTonic ya que posiblemente aumente su cartera de clientes y por otro lado el autor de este proyecto, ya que habré podido formarme a fondo sobre tecnologías punteras utilizadas en este proyecto como se describirán a posteriori.

### 1.4.1. OBJETIVOS DEL PROYECTO

Una vez descrito el contexto sobre el cual será realizado el proyecto, es decir, describir las principales funcionalidades de la herramienta de ServiceTonic, el siguiente paso es describir el objetivo general del mismo.

Como he descrito en el apartado *1.1. Contexto y actores*, una de las mejores más demandadas por los clientes es la de poder geolocalizar las diferentes funcionalidades del software de ServiceTonic. Ahí es donde entra en juego este proyecto, en el desarrollo de una mejora para localizar los activos de una empresa.

Entonces, definiría como uno de los objetivos primordiales de este proyecto desarrollar un sistema de geolocalización de activos basado sobre varias premisas:

En primer lugar, ha de ser lo más usable posible. Considero que el software de ServiceTonic, aunque sea sencillo de utilizar, es bastante complejo y requiere un proceso de aprendizaje antes de ponerlo en producción en la empresa. Se pretende que sea fácil de utilizar pero que el cliente tenga toda la libertad posible para personalizar esta mejora a su gusto mediante diferentes parámetros de configuración.

En segundo lugar, la privacidad es un aspecto muy importante y muy delicado. Estamos hablando de que una de las funcionalidades de la mejora será la de geolocalizar a los agentes de una empresa. Por lo tanto, debemos asegurarnos de que el usuario final tenga el control total sobre si está siendo geolocalizado por el software o saber a qué agentes se está geolocalizando.

En tercer lugar, algo más relacionado con los asuntos internos de la empresa y sus métodos de trabajo: seguir unas pautas preestablecidas y un orden a la hora de desarrollar el código de la mejora. El código desarrollado en este proyecto ha de ser lo más comprensible posible por si lo tuviera que seguir otra persona. Y ya no sólo por este motivo, sino por objetivos personales, para ir aprendiendo buenas mecánicas para hacer un código comprensible y ordenado.

#### 1.4.1.1. OBJETIVOS PERSONALES

Así pues, una vez comentado el objetivo primordial del proyecto de cara a la empresa, me he propuesto cumplir varios objetivos personales en el transcurso del proyecto:

El primer objetivo es el de realizar un buen proyecto. Por simple que suene es el objetivo más claro que toda persona debería querer alcanzar. Esto conlleva, a parte de las premisas comentadas anteriormente, documentar apropiadamente todo el proyecto, hacer una buena presentación, entre otros aspectos.

El segundo objetivo, ya comentado anteriormente, es el de obtener buenas prácticas o mecánicas a la hora de desarrollar código. Es una meta que me parece realmente complicada ya que, bajo mi punto de vista, es más complejo desarrollar un código que entiendan otros desarrolladores que no un código que funcione.

Otro objetivo más secundario, sería el de ampliar mis conocimientos de programación a partir de los lenguajes con los que se desarrollará este proyecto.

En conclusión, si tuviera que agrupar todos los objetivos en uno sólo diría que el objetivo principal de este proyecto es aprender. Establecer contacto con lo que es desarrollar un proyecto complejo, utilizar más de una rama que no sea la de master, haber de entender la forma de trabajar de otras personas, adaptarte a ella.

## 1.5. METODOLOGÍA Y RIGOR

### 1.5.1. METODOLOGÍA AGILE

En ServiceTonic, se funciona a través de la metodología Agile.

Esta metodología consiste en definir en periodos de dos semanas aproximadamente el trabajo a realizar. Este periodo de dos semanas se denomina *sprint*. Al final de cada *sprint* se hace un *sprint review* donde se ven los resultados de ese periodo de tiempo y se concretan los objetivos para el siguiente *sprint*. Por otro lado, se realizan pequeñas reuniones, llamadas *daily,* de diez minutos justo al empezar la jornada laboral para definir que se realizará ese día, que se realizó el día anterior y qué problemas se produjeron.

En esta metodología la empresa se divide en diferentes equipos según su funcionalidad. Cada equipo tiene su *Scrum Master* que es el encargado de coordinar su equipo, repartir las tareas a cada integrante del equipo y responsable del correcto funcionamiento del equipo.

La metodología Agile se rige mediante el *Manifiesto Agile*[1] se rige por doce principios los cuales se aplican al día a día en ServiceTonic:

1. Satisfacción del cliente.
2. Bienvenidos los nuevos requisitos.
3. Entregas por semanas.
4. Posibilidad de medir el progreso.
5. Desarrollo sostenible.
6. Trabajo cercano.
7. Conversación cara a cara.
8. Motivación y confianza.  Excelencia técnica y buen diseño.
9. Simplicidad.
10. Autogestión de los equipos.
11. Adaptación circunstancias cambiantes.

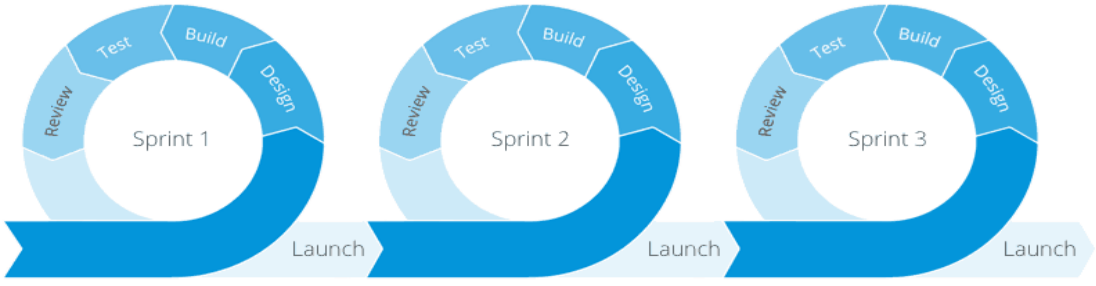


*Figura 1.5.1.1. Proceso seguido en la metodología Agile*

Las herramientas para realizar el seguimiento han sido básicamente las establecidas en la metodología Agile. En este caso, la herramienta principal ha sido un *Kanban*[2]. Un *Kanban* es una

pizarra utilizada en las metodologías de tipo SCRUM para hacer el seguimiento del desarrollo de un producto. El aspecto que tiene es el siguiente:

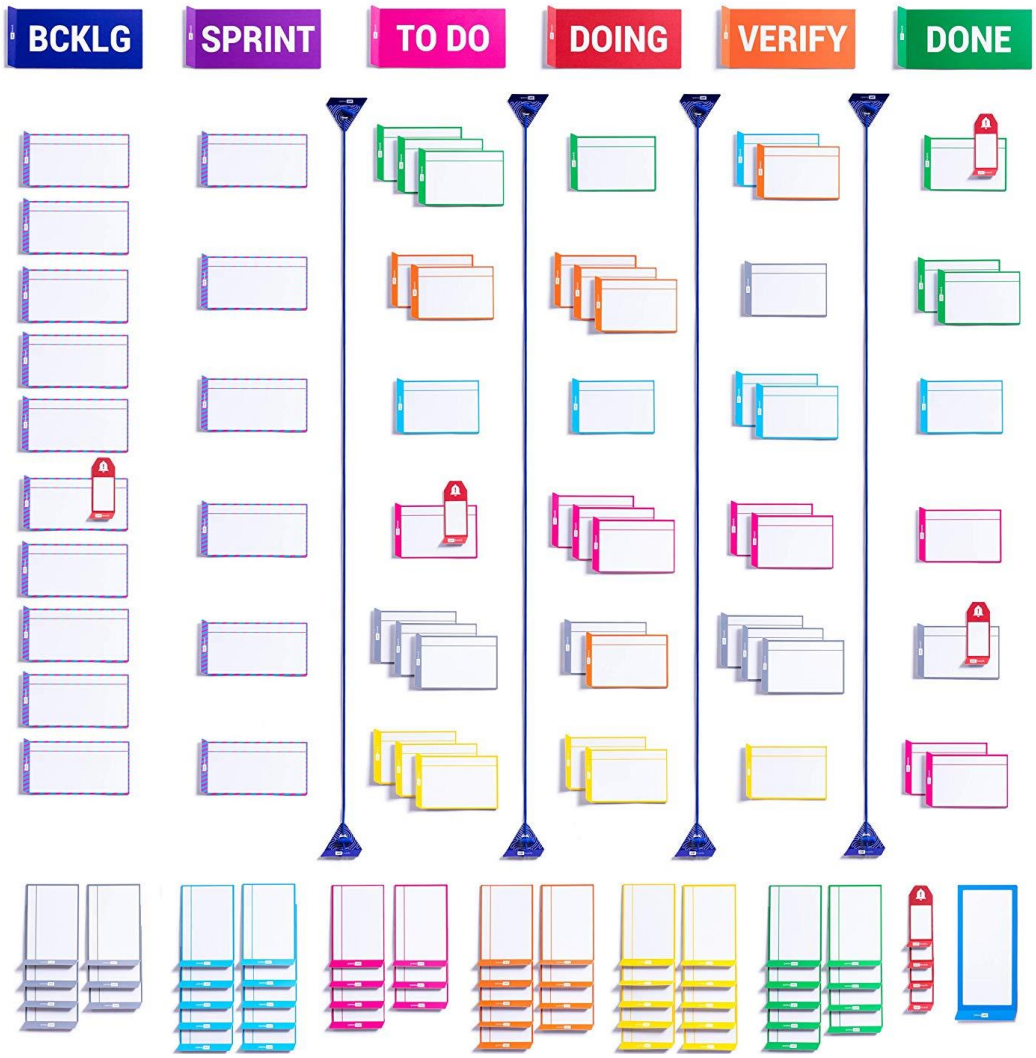


*Figura 1.5.1.2. Aspecto de la pizarra Kanban*

La pizarra está dividida en seis apartados los cuáles los forman:

1. ***Backlog***: Son las mejoras propuestas en las reuniones realizadas pero que aun no están en desarrollo.
2. ***Sprint***: Son cada una de las tarjetas de las mejoras que se están desarrollando en ese momento las cuales se subdividen en pequeñas mejoras.
3. ***To do***: Son aquellas pequeñas tareas que forman parte de una tarjeta del apartado anterior que están pendientes de desarrollar.
4. ***Doing***: Tareas que están en desarrollo.
5. ***Verify***: Tareas que están desarrolladas pero que están pendientes de revisar por el *Scrum Master*.
6. ***Done***: Tareas desarrolladas, revisadas y testeadas y, por tanto, finalizadas.

### 1.5.2. PROBLEMAS O OBSTÁCULOS CON LA METODOLOGÍA AGILE

Si es cierto también por otro lado, que durante el desarrollo del proyecto hemos topado con algún problema o obstáculo con la metodología.

En primer lugar, esta metodología requiere dedicación por parte de los integrantes del equipo para que el tablero esté actualizado en cada momento y que el *Scrum Master* se responsabilice de que los integrantes realicen esa tarea. Debido a la carga de trabajo que ha supuesto ya sea para mí este proyecto, o para el *Scrum Master*, es decir, el director de este proyecto, las diferentes mejoras ajenas a este proyecto o para los demás integrantes del equipo estas mismas, ha habido etapas del proyecto que quizás el tablero no estaba actualizado del todo.

En segundo lugar, en cuanto a las reuniones marcadas por la metodología que se realizan cada dos semanas no han sido estrictamente regulares. Esto es debido, a que no siempre las predicciones que se realizan en estas mismas son exactas. Por lo tanto, ha habido reuniones que quizá se retrasaban una semana para que tuviésemos una mejora realizada consistente que explicar a los demás integrantes del equipo. Así pues, estos periodos de dos semanas definidos por la metodología Agile han sido de tres semanas en ocasiones debido a ese motivo.

Finalmente, en cuanto las reuniones *daily*, en mi opinión veo que no son tan útiles como se define en esta metodología debido a que en muchas de ellas prácticamente se estaba realizando la misma tarea que al día anterior y por otro lado, no había surgido ningún problema o algún problema que pude solucionar por mí mismo.

No obstante, en el desarrollo del proyecto no se ha producido ningún cambio de metodología ya que pese a los problemas surgidos a partir de ella hemos visto que el funcionamiento ha sido ágil y organizado.

### 1.5.3. LA METODOLOGÍA AGILE APLICADA AL PROYECTO

En conclusión, a grandes rasgos la metodología que la empresa me ha hecho seguir es el siguiente:

Cuando se han definido los objetivos del próximo periodo de dos semanas comienzo con el desarrollo del código. En el caso de que surjan dudas las comento con el director del proyecto, el *Scrum Master*, y se intentan solucionar lo más rápido y mejor posible. Una vez acabo con el desarrollo del código comienzo el *testing* y cuando funciona todo debidamente el director del proyecto procede a su validación. Si el código es dado del visto bueno, comienzo a documentar los pasos realizados y su funcionamiento.

Una vez se completan los pasos de desarrollo de código, *testing*, validación y documentación se

---

1. En el año 2001, se reunieron los CEOs de las principales empresas de 'software' en Utah. Allí pusieron en común las mejores prácticas de cada compañía y crearon el 'Manifiesto Agile'.
2. En japonés, *kanban* se traduce literalmente como "señal visual".

definen nuevos pasos a seguir y se reinicia el ciclo de Agile mostrado en la figura 1.5.1.1.

En mi caso concreto, una vez terminaba, ya fuera una tarjeta del apartado Sprint del *Kanban* o una tarjeta de una parte de la mejora pasaba al estado de *Verify* y el director del proyecto pasaba a verificar que la mejora cumplía todos los requisitos especificados o tenía que aplicarse alguna corrección.

Para llegar a la solución propuesta de cada *Sprint*, se debate entre el director del proyecto y yo cómo se debería desarrollar esa mejora que se realizará en el próximo periodo aportando ambos diferentes puntos de vista y prevaleciendo la decisión del director del proyecto.

## 1.6. PLANIFICACIÓN DEL PROYECTO

### 1.6.1. DATOS GLOBALES

El proyecto se inició el día 14 de noviembre de 2018, el día que entré en la empresa ServiceTonic. El proyecto acabó el día 30 de junio para su posterior presentación el 4 de julio de 2019. En total, el trabajo realizado durante el proyecto ha sido de 735 horas a razón de 5 horas al día de lunes a viernes.

### 1.6.2. FASES DEL PROYECTO (PLANIFICACIÓN PRELIMINAR)

El proyecto lo dividiría en tres fases[1] principales:

1. **Fase de aprendizaje** (noviembre 2018 – enero 2019): Fase donde se aprende y se dan los primeros pasos con las tecnologías que se usaran durante el proyecto. Una vez se está familiarizado con la tecnología se empieza a comprender cómo está montado, organizado y cómo funciona el *software* de ServiceTonic.

2. **Fase de desarrollo** (enero 2019 – junio 2019): Fase principal del proyecto donde se desarrolla el proyecto en sí. Esta fase esta subdividida en 6 subfases[2]:
   - 2.1. Geolocalizar contactos.
     - 2.1.1. Diseño.
     - 2.1.2. Desarrollo.
     - 2.1.3. *Testing*.
     - 2.1.4. *Sprint review.*
   - 2.2. Geolocalizar agentes automáticamente.
     - 2.2.1.Diseño.
     - 2.2.2. Desarrollo.
     - 2.2.3. *Testing*.
     - 2.2.4. *Sprint review.*
   - 2.3. Geolocalizar CIs.
     - 2.3.1.Diseño.
     - 2.3.2. Desarrollo.
     - 2.3.3. *Testing*.
     - 2.3.4. *Sprint review.*
   - 2.4. Check-in – Check-out.
     - 2.4.1.Diseño.
     - 2.4.2. Desarrollo.
     - 2.4.3. *Testing*.
     - 2.4.4. *Sprint review.*

1. Véase el apartado 6. Anexos donde se muestra el diagrama de Gantt del proyecto.
2. Véase el apartado *2. La mejora* donde se explica a fondo en qué consistía cada subfase.

2.5. Geolocalizar tickets y problemas.
  2.5.1. Diseño.
  2.5.2. Desarrollo.
  2.5.3. *Testing*.
  2.5.4. *Sprint review.*
2.6. Explotación de la información.
  2.6.1. Diseño.
  2.6.2. Desarrollo.
  2.6.3. *Testing*.
  2.6.4. *Sprint review.*

3. **Fase de documentación** (junio 2019): Fase final del proyecto donde se realiza la documentación y memoria de la totalidad del proyecto especificando todo el progreso y trabajo realizado.

### 1.6.3. ESTIMACIÓN PRELIMINAR LIMITACIONES Y RIESGOS DEL TFG

Existen algunas limitaciones y riesgos respecto al desarrollo del proyecto:

En primer lugar, la limitación más previsible y notable que íbamos a tener durante la realización del proyecto era la dependencia de servicios externos como librerías o, en este caso, la API de Google *Maps*. Se ha tenido que adaptar el código para optimizar y reducir las llamadas a la API de Google *Maps* debido a que no son gratuitas, pasado un cierto número de llamadas.

Por otro lado, uno de los riesgos más previsibles era, continuando con la limitación del apartado anterior, tener que rehacer código debido a que una librería no ofrecía o no funcionaba como esperábamos.

Y finalmente, uno de los riesgos más secundarios y remotos era el de no poder finalizar el proyecto a tiempo. Este riesgo era el de menos importancia debido a la modularidad del proyecto, es decir, el proyecto se divide en geolocalizar varias funcionalidades o ítems. En caso de que no hubiera tiempo de finalizar a tiempo el proyecto, se geolocalizarían *tickets*, contactos y CIs y no agentes, por poner un ejemplo.

### 1.6.4. REVISIÓN COMPROMISOS ANTERIORES

Tanto objetivos como alcance como costes del proyecto se han mantenido como al inicio del proyecto. El único compromiso que ha cambiado respecto al inicio ha sido la planificación mostrada en este apartado.

Como era de esperar, hemos tenido que extender los diferentes plazos en los que estaban marcadas las diferentes fases del proyecto, debido a problemas que iban surgiendo a medida que se iban desarrollando las diferentes funcionalidades.

Por otro lado, comentar que en cada fase se iban revisando y corrigiendo los fallos[1] que iban saliendo de las anteriores. Es decir, una vez que se daba por acabada una fase no quería decir que estuviera preparada para su lanzamiento y que no se fuera a modificar o corregir en un futuro, ya que no hay ninguna fase que sea independiente de ninguna otra, por lo que una mejora depende de que todas las demás funcionen correctamente y a la inversa.

Así que la justificación de los cambios realizados en la planificación del proyecto se basa en la corrección de errores para sacar al mercado un producto fiable, seguro, funcional y usable.

1. Para ver los obstáculos o problemas del proyecto véase el apartado *3. Conclusiones, problemas y obstáculos*.

## 1.7. GESTIÓN ECONÓMICA DEL PROYECTO

En este apartado se hará una valoración económica del proyecto teniendo en cuenta todos los aspectos posibles. Comentar el presupuesto está dividido en la parte fija, tanto recursos *hardware*, como humanos, como energéticos, y por otro lado la parte variable.

El motivo de la parte variable de la valoración económica es el uso e integración de la API de Google *Maps* al proyecto. Como se ha comentado en apartados anteriores, el precio de la API de Google *Maps* va dictaminado por el precio por uso que se le dé en el negocio. Se parte de unos créditos iniciales gratuitos cada mes, con esos créditos podemos hacer un número limitado de llamadas a la API, una vez se sobrepase ese límite se empieza a aplicar una tarifa de pago por uso. Debido a esa variabilidad de la tarifa y a que inicialmente no se sabe cuántos clientes harán uso de la mejora no se puede determinar con exactitud el importe que supondrá el uso de la API de Google *Maps*.

Una vez comentado eso, pasamos a determinar el presupuesto del proyecto desglosado como se ha comentado:

| Recursos físicos | |
|---|---|
| Ítem | Precio |
| PC Acer Aspire XC-705 Intel i5-4460/4GB/1TB/GT720 | 509 € |
| Teclado + Ratón | 15 € |
| Monitor LED 24” S22F350 | 112 € |
| Monitor LG L1752s | 23 € |
| | |
| **Recursos humanos** | |
| Ítem | Precio |
| Desarrollador | 6615€[1] |
| | |
| **Recursos energéticos** | |
| Ítem | Precio |
| Consumo PC (~100W) | 10’86€[2] |
| | |
| **Recursos variables** | |
| Ítem | Precio |
| API Google *Maps* | - |

En total, el presupuesto teniendo sólo en cuenta la parte fija sería de **7284,86 €.**

La viabilidad del proyecto establecida por la empresa es positiva debida a la demanda de esta funcionalidad por parte de los clientes. Uno de los aspectos que sí se tiene claro, como se ha comentado en apartados previos es que el uso de este proyecto supondrá un coste adicional en la licencia que las empresas tienen contratada. Las empresas tendrán la posibilidad, eso sí, de elegir si deciden incorporar esta funcionalidad a su licencia, suponiendo un suplemento adicional, o no, donde seguirán pagando el mismo importe por la licencia.

1. 6615 € a razón de 9€/hora por un total de 735 horas de la duración del proyecto
2. El precio del kWh utilizado para el cálculo del precio total ha sido de 0,147756€.

## 1.8. SOSTENIBILIDAD Y COMPROMISO SOCIAL

### 1.8.1. MATRIZ DE SOSTENIBILIDAD

En este apartado se responderá a las preguntas formuladas por la matriz de sostenibilidad mostrada abajo y se lanzarán unas conclusiones a partir de las respuestas generadas para ver cuán sostenible es este proyecto:

| | | PPP | Vida Útil | Riesgos |
|---|---|---|---|---|
| Ambiental | I | ¿Has estimado el impacto ambiental que tendrá la realización del proyecto? ¿Te has planteado minimizar el impacto, por ejemplo, reutilizando recursos? | ¿Cómo se resuelve actualmente el problema que quieres abordar (estado del arte)? ¿En qué mejorará ambientalmente tu solución a las existentes? | |
| | F | ¿Has cuantificado el impacto ambiental de la realización del proyecto? ¿Qué medidas has tomado para reducir el impacto? ¿Has cuantificado esta reducción? | ¿Qué recursos estimas que se usarán durante la vida útil del proyecto? ¿Cuál será el impacto ambiental de estos recursos? | ¿Podrían producirse escenarios que hiciesen aumentar la huella ecológica del proyecto? |
| | | Si hicieras de nuevo el proyecto, ¿podrías realizarlo con menos recursos? | ¿El proyecto permitirá reducir el uso de otros recursos? ¿Globalmente, el uso del proyecto mejorará o empeorará la huella ecológica? | |
| Económico | I | ¿Has estimado el coste de la realización del proyecto (recursos humanos y materiales)? | ¿Cómo se resuelve actualmente el problema que quieres abordar (estado del arte)? ¿En qué mejorará económicamente tu solución a las existentes? | |
| | F | ¿Has cuantificado el coste (recursos humanos y materiales) de la realización del proyecto? ¿Qué decisiones has tomado para reducir el coste? ¿Has cuantificado este ahorro? | ¿Qué coste estimas que tendrá el proyecto durante su vida útil? ¿Se podría reducir este coste para hacerlo más viable? | ¿Podrían producirse escenarios que perjudicasen la viabilidad del proyecto? |
| | | ¿Se ha ajustado el coste previsto al coste final? ¿Has justificado las diferencias (lecciones aprendidas)? | ¿Se ha tenido en cuenta el coste de los ajustes/actualizaciones/reparaciones durante la vida útil del proyecto? | |
| Social | I | ¿Qué crees que te va a aportar a nivel personal la realización de este proyecto? | ¿Cómo se resuelve actualmente el problema que quieres abordar (estado del arte)? ¿En qué mejorará socialmente (calidad de vida) tu solución a las existentes? | |
| | | | ¿Existe una necesidad real del proyecto? | |
| | F | ¿La realización de este proyecto ha implicado reflexiones significativas a nivel personal, profesional o ético de las personas que han intervenido? | ¿Quién se beneficiará del uso del proyecto? ¿Hay algún colectivo que puede verse perjudicado por el proyecto? ¿En qué medida? | ¿Podrían producirse escenarios que hiciesen que el proyecto fuese perjudicial para algún segmento particular de la población? |
| | | | ¿En qué medida soluciona el proyecto el problema planteado inicialmente? | ¿Podría crear el proyecto algún tipo de dependencia que dejase a los usuarios en posición de debilidad? |

*Figura 5.1. Matriz de sostenibilidad de un proyecto*

**¿Has estimado el impacto ambiental que tendrá la realización del proyecto? ¿Te has planteado minimizar el impacto, por ejemplo, reutilizando recursos?**

Realmente no he parado a estimar el impacto ambiental que tendrá el proyecto ya que considero que es nulo, por lo menos directamente. Indirectamente, pues el consumo energético de equipo donde se desarrolla el código. En términos de reutilización de recursos, los únicos recursos que se reutilizarían serían recursos software ya que se procura que el código desarrollado sea lo más reutilizable posible.

**¿Cómo se resuelve actualmente el problema que quieres abordar? ¿En qué mejorará ambientalmente tu solución a las existentes?**

Se resuelve mediante la integración de un servicio de geolocalización externo como es Google *Maps*. En nada, básicamente estamos añadiendo una funcionalidad muy común a un software muy personalizada.

¿Has cuantificado el impacto ambiental de la realización del proyecto? ¿Qué medidas has tomado para reducir el impacto?

Ni he cuantificado el impacto ni he tomado ninguna medida para reducir el impacto que sean relevantes para reducir el impacto más allá de apagar el equipo donde se desarrolla el código para no consumir energía de más.

Si hicieras el mismo proyecto de nuevo, ¿podrías realizarlo con menos recursos?

Solamente se podría reducir el tiempo llevado a cabo ya que sabría qué pasos seguir y cómo realizarlo. Por lo demás, los recursos seguirían siendo los mismos.

¿Qué recursos estimas que se usarán durante la vida útil del proyecto? ¿Cuál será el impacto medioambiental de estos recursos?

Electricidad para el mantenimiento de los servidores. Pues el impacto que genera la producción de electricidad.

¿El proyecto permitirá reducir el uso de otros recursos? ¿Globalmente, el uso del proyecto mejorará o empeorará la huella ecológica?

No.

¿Podrían producirse escenarios que hiciesen aumentar la huella ecológica del proyecto?

No depende de nosotros.

¿Has estimado el coste de la realización del proyecto (recursos humanos y materiales)?

No. Se justifica la respuesta en las siguientes preguntas.

¿Has cuantificado el coste (recursos humanos y materiales) de la realización del proyecto? ¿Qué decisiones has tomado para reducir el coste? ¿Has cuantificado este ahorro?

Ninguno más allá del importe de la beca que percibiré por la realización del proyecto. Los demás recursos son reutilizados.

¿Se ha ajustado el coste previsto al coste final? ¿Has justificado las diferencias (lecciones aprendidas)?

De momento no ya que hasta que no entre a producción no se podrá saber con exactitud. El coste del uso de la API de Google *Maps* es variable y depende de los clientes que la utilicen ya que se utiliza como modelo de negocio en la API el pago por uso.

¿Cómo se resuelve actualmente el problema que se quiere abordar (estado del arte)? ¿En qué mejorará económicamente tu solución a las existentes?

Mediante el uso de la API de Google *Maps*. En nada ya que básicamente añadimos una funcionalidad común a un software en concreto como es el de ServiceTonic.

**¿Qué coste estimas que tendrá el proyecto durante su vida útil? ¿Se podría reducir este coste para hacerlo más viable?**

Es un tema que aún está pendiente ya que dependemos de las tarifas de pago por uso de la API de Google *Maps*.

**¿Se ha tenido en cuenta el coste de los ajustes / actualizaciones / reparaciones durante la vida útil del proyecto?**

Ese coste para la empresa sería el de mantener a un empleado, tanto a mí como a otra persona dispuesta a mantener el proyecto, una vez terminado el proyecto.

**¿Podrían reducirse escenarios que perjudicasen la viabilidad del proyecto?**

Seguramente aunque se ha intentado evitar en todo momento y que sea lo más viable posible.

**¿Qué crees que te va a aportar a nivel personal la realización de este proyecto?**

Experiencia.

**¿La realización de este proyecto ha implicado reflexiones significativas a nivel personal, profesional o ético de las personas que han intervenido?**

Ninguna más allá de cómo llevar el desarrollo de la herramienta y las posibles dudas o problemas que hayan podido surgir durante el proyecto, tanto a mi como al director del proyecto.

**¿Quién se beneficiará del uso del proyecto? ¿Hay algún colectivo que puede verse perjudicado por el proyecto? ¿En qué medida?**

Directamente la empresa ya que será quien se lucre a través del proyecto e indirectamente las empresas clientas de ServiceTonic que contraten esta funcionalidad ya que tendrán control de la posición de cada activo de su empresa.

**¿En qué medida soluciona el proyecto el problema planteado inicialmente?**

El objetivo es que solucione por completo el problema que se planteó inicialmente, geolocalizar los diferentes activos de ServiceTonic.

**¿Podrían producirse escenarios que hiciesen que el proyecto fuese perjudicial para algún segmento particular de la población?**

Directamente no pero el tema de ser localizado en segundo plano puede llevar a producir algunos malentendidos entre empresa y agente, aunque la herramienta es totalmente configurable y el agente sabrá en todo momento si está siendo geolocalizado o no.

**¿Podría crear el proyecto algún tipo de dependencia que dejase a los usuarios en posición de debilidad?**

Esta herramienta es una mejora que se incluirá en un servicio mayor y no está pensado ni se espera que crea ninguna dependencia.

### 1.8.2. ANÁLISIS DE RIESGOS

En este apartado se presentarán los riesgos, tanto sociales, ambientales como económicos.

En primer lugar, como riesgos ambientales, no es que sean notorios debido a que la implementación de la mejora no supondrá ninguna contratación adicional de servidores o subida de la tarifa del consumo eléctrico. Por otro lado, tampoco tendrá, como se ha comentado en las cuestiones de la matriz de sostenibilidad ningún impacto sobre el medio ambiente.

Siguiendo con los riesgos sociales, en un principio se creerían que iban a ser mayores debido a la implementación de la geolocalización automática de agentes, ya que es una funcionalidad que determina tu geoposición en segundo plano. Debido a que finalmente se optó por no desarrollarla[1], finalmente los riesgos sociales se reducen bastante y se resumen a que en ciertos momentos el usuario ha de introducir su geoposición al sistema.

Finalmente, como riesgos económicos cabe la remota posibilidad de que la mejora que implementa este proyecto no sea utilizada por la cantidad de clientes esperada a pesar del *feedback* obtenido por ellos mismos reclamando esa mejora. Por otro lado, cabe la posibilidad de errores de cálculo relacionados con las tarifas de pago por uso de la API de Google *Maps* en un futuro posterior a este proyecto calculando el importe. Este último riesgo no se podrá saber hasta que se lancé esta mejora y se observe el número de clientes que la utilicen y qué uso le dan.

### 1.8.3. CONCLUSIONES SOBRE LA SOSTENIBILIDAD DEL PROYECTO

Las conclusiones personales que saco después de haber realizado el análisis de riesgos y un informe de sostenibilidad a partir de la matriz de sostenibilidad son las siguientes:

Mi opinión personal sobre la sostenibilidad de este proyecto es en general muy buena debido a que la gran mayoría de código de lo que será esta funcionalidad está desarrollado, como forma parte este proyecto. Faltarían las mejoras propuestas por los clientes una vez hecho el *release* de la funcionalidad descrita en este proyecto.

Se ha pensado en todo momento del proyecto que el código fuera reutilizable, entendible y fácil de aplicar cualquier mejora descrita por los clientes.

Por otro lado, queda la incógnita de la sostenibilidad económica del proyecto debido al número de clientes que contratarán esta mejora. De todas formas, el hecho de que el importe del uso de la API de Google *Maps* sea incremental conforme aumenten el número de llamadas a la API hace que la sostenibilidad económica sea favorable.

1. Véase el apartado 3. Conclusiones, problemas y obstáculos para conocer el motivo por el cual se optó por no desarrollar esa parte de la mejora.

## 1.9. TECNOLOGÍAS USADAS EN EL PROYECTO

### 1.9.1. INTRODUCCIÓN

En primer lugar, ServiceTonic es una herramienta web, como ya se ha comentado en apartados anteriores. Dicho esto, el proyecto de ServiceTonic hecho con la tecnología de *Spring Boot*, una tecnología utilizada para crear de manera sencilla aplicaciones *stand-alone*. Esta tecnología permite correr la aplicación bajo un servidor web, en este caso, la aplicación corre bajo un servidor *Apache Tomcat*.

El proyecto utiliza la tecnología de gestión de dependencias de *Maven* para especificar que librerías externas se apoyará el proyecto.

El *software* está escrito en varios lenguajes de programación, recordemos que es una aplicación 100% web. Aunque sea una aplicación web, se habrá de realizar la instalación en el equipo, en el caso de que estemos en PC, ya que se instalan diferentes componentes que comentaremos a continuación.

El principal lenguaje de programación con el que está desarrollado ServiceTonic es Java. Concretamente, bajo el *framework* de Java Server Faces que es utilizado para el desarrollo de aplicaciones web. La aplicación hace uso de una biblioteca de componentes para JSF llamada *Primefaces*. ServiceTonic, pues, está desarrollado a partir de una plantilla con un estilo predefinido proporcionada por *Primefaces*.

Para almacenar todos los datos de la aplicación se hace uso de un gestor de bases de datos. ServiceTonic, en la instalación, da la opción de ser instalado en los gestores de bases de datos MySQL, Oracle o SQL Server.

El medio de desarrollo más concreto pero principal que da vida a este proyecto es la API de Google *Maps*. Debido a que el uso que se le va a dar uso es en un entorno empresarial, ha de realizarse un estudio para ver qué costes supone a la empresa y mecanismos para limitar las peticiones a la API por parte de los clientes según la licencia que tengan contratada.

Por último, como medios de desarrollo más secundarios formarían parte el IDE donde se desarrolla el código en ServiceTonic, Eclipse, el servidor de Git local, GitLab, y Jenkins para integración contínua, entre otros.

### 1.9.2. JAVA, JAVA SERVER FACES Y PRIMEFACES

En este apartado, explicaré el funcionamiento de Java y, principalmente, de uno de sus *frameworks* utilizados en este proyecto, Java Server Faces.


*Figura 1.8.2.1. Java, uno de los lenguajes de programación del desarrollo de ServiceTonic*

Java Server Faces utiliza el modelo MVC (Modelo – Vista – Controlador) para gestionar la aplicación web.

En primer lugar, el modelo lo formarían las diferentes clases Java que en Java Server Faces son los denominados *Managed Beans*, esta clase y sus atributos puede ser consultada por la vista mediante una anotación concreta. Asimismo, si en la vista se definen las acciones sobre los componentes, en el controlador se definen las acciones sobre la aplicación tales como realizar cálculos, consultar el modelo de datos y definir parámetros.

Continuando con el funcionamiento de Java Server Faces, haré un breve resumen del funcionamiento de éste de forma generalizada:

El código fuente de la página a la accedemos de ServiceTonic está formada por dos elementos: el primero, la declaración de la interfaz de usuario con Java Server Faces. El formato de esta interfaz de usuario es realizado mediante una adaptación de la anotación XML, XHTML la cuál es utilizada en Java Server Faces. En esta declaración se describen los componentes que formarán la interfaz como puedan ser tablas, imágenes, botones, métodos de entrada de datos del usuario, etc. El segundo elemento, son los datos introducidos en estos componentes declarados en la interfaz de usuario.

Para que estos dos elementos formen el código fuente, es decir, la página *html* que será el resultado final, antes deben pasar por el ciclo de vida de Java Server Faces. Las fases de este son:

1. Restaurar la vista: Se obtiene el árbol de componentes de la vista JSF de la petición a partir del código fuente de la página. Si se había generado antes se recupera y si no, es decir, se visita la página por primera vez, se genera a partir de la declaración de los componentes de la interfaz de usuario a partir de la página XHTML.
2. Aplicar los valores de la petición: Cuando se haya obtenido el árbol de componentes de la vista JSF, se procesan todos los valores de este y se convierten todos los datos de la petición a tipos de datos Java.
3. Procesar las validaciones: Se validan todos los datos obtenidos en el paso anterior. En el caso de que haya surgido algún error en este paso se encola un mensaje de error y concluye el ciclo de vida, saltando al último paso.
4. Actualizar los valores del modelo: Se actualizan las propiedades del modelo de datos definido en cada *Managed Bean* gestionada asociada a cada componente.
5. Renderizar la respuesta: Se renderiza el resultado final del ciclo JSF.

Al final de cada una de las fases, se comprueba si hay algún evento que debe ser procesado en esa fase en concreto, como, por ejemplo, una llamada *Ajax*[1].

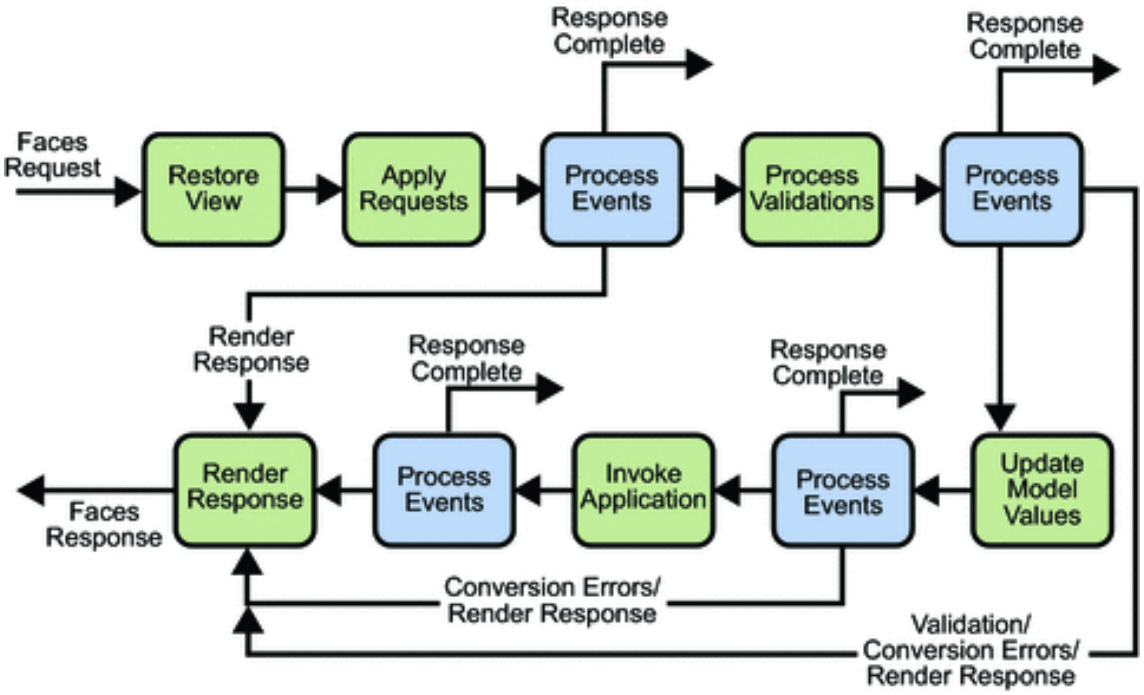


*Figura 1.8.2.2. Ciclo de vida de Java Server Faces.*

### 1.9.3. HIBERNATE, MYSQL, SQL SERVER Y ORACLE

Para el mapeo de los objetos de las clases Java del modelo de Java en las correspondientes tablas de la base de datos se ha utilizado la herramienta *Hibernate* la cual lo hace posible mediante anotaciones sobre el modelo de clases Java.

```
@Temporal(TemporalType.TIMESTAMP)
@Column(name = "LAST_UPDATE_DATE", nullable = false, length = 19)
public Date getLastUpdateDate() {
    return this.lastUpdateDate;
}

public void setLastUpdateDate(Date lastUpdateDate) {
    this.lastUpdateDate = lastUpdateDate;
}
```

*Figura 1.8.3.1. Mapeo de un atributo de una clase Java del modelo de datos a una tabla de la base de datos.*

1. AJAX son las siglas de Asynchronous JavaScript And Xml, (JavaScript asíncrono y XML), un conjunto de tecnologías que permiten actualizar contenidos web sin tener que volver a cargar la página.

Por otro lado, *Hibernate* utiliza un fichero aparte donde se establecen los diferentes parámetros de conexión como la dirección de la base de datos, el tipo, las credenciales, el puerto, etc.


*Figura 1.8.3.2. Logo de Hibernate, la herramienta utilizada en el proyecto para el mapeo del modelo Java al modelo de base de datos.*

Finalmente, antes del desarrollo de cada mejora se realizan los scripts SQL para MySQL, SQL Server y Oracle para hacer la modificación de la base de datos añadiendo nuevas tablas, registros o modificaciones.




*Figura 1.8.3.3. Logos de MySQL, SQL Server y Oracle, las bases de datos utilizadas en ServiceTonic*.

### 1.9.4. GOOGLE MAPS API

La integración de la API de Google *Maps* es bastante sencilla. Una vez hemos contratado el plan deseado que se ajuste a las necesidades del negocio se obtiene una *key* y con esta se realizan las peticiones a la API. Para añadirla al proyecto se utiliza la etiqueta *script* y como fuente del script se indica la URL '*https://maps.google.com/maps/api/js?key='* seguida de la *key* obtenida.

# 2. LA MEJORA

## 2.1. INTRODUCCIÓN

En este capítulo se explicaran las diferentes funcionalidades del proyecto en sí. Aportando, en un primer lugar, en este capítulo, una visión generalizada de ellas para tener una primera idea del funcionamiento de estas.

Como se ha comentado previamente en el apartado *1.6. Planificación del proyecto* el proyecto está divido en diferentes fase las cuáles se explicarán en los siguientes capítulos.

Cada fase comienza por la lectura y comprensión del documento, redactado por el director del proyecto, donde se explican los requisitos que cada funcionalidad ha de cumplir. Una vez finalizado ese paso, se procede con la creación del *script* de modificación de la base de datos. En cada funcionalidad, se necesita alterar la estructura de la base de datos añadiendo ya sean nuevas tablas o nuevas columnas a tablas existentes. Cada sentencia del script debe estar disponible tanto para bases de datos MySQL, Oracle y SQL Server por lo que cada sentencia tiene 2 más equivalentes en otros lenguajes. Al final de la creación, validación de cualquier funcionalidad se añade el contenido de este *script* a otro *script* donde éste será el que entre en producción junto a las demás funcionalidades ajenas a este proyecto en la versión del *release* donde se incorpore la mejora de este proyecto.

```
mysql ALTER TABLE configuration_item ADD COLUMN ST_ADDRESS varchar(250);
mysql ALTER TABLE configuration_item ADD COLUMN ST_LATITUDE varchar(25);
mysql ALTER TABLE configuration_item ADD COLUMN ST_LONGITUDE varchar(25);
sqlserver ALTER TABLE configuration_item ADD ST_ADDRESS varchar(250);
sqlserver ALTER TABLE configuration_item ADD ST_LATITUDE varchar(25);
sqlserver ALTER TABLE configuration_item ADD ST_LONGITUDE varchar(25);
oracle ALTER TABLE configuration_item ADD ST_ADDRESS varchar(250);
oracle ALTER TABLE configuration_item ADD ST_LATITUDE varchar(25);
oracle ALTER TABLE configuration_item ADD ST_LONGITUDE varchar(25);
```

*Figura 2.1.1. Ejemplo de sentencias de una funcionalidad del proyecto.*

Una vez la tarea anterior se realiza, siempre empiezo por construir la vista de la funcionalidad ya que ésta dependerá del comportamiento que se añadirá posteriormente. A medida que se va avanzando con el proyecto, cabe remarcar que cada vez el trabajo necesario y empeñado en la parte visual es menor, debido a que se hacen uso de componentes reutilizables como son los denominados en Java Server Faces, *composites*[1].

1. Un *composite* es un componente de JSF que es un tipo especial de plantilla la cuál es, esencialmente, un trozo de código reutilizable que se comporta de una forma concreta.

Todos elementos del proyecto, para ser geolocalizados, pasan por el mismo componente[1] explicado en el párrafo anterior. Es por este motivo la importancia que obtiene en el proyecto la reusabilidad del código.

Finalmente, el proceso seguido para guardar los diferentes registros en la base de datos es bastante similar para todas las funcionalidades. Gracias a la librería de *Hibernate* podemos mapear una clase Java mediante anotaciones, como se ha visto en el apartado *1.9.3. Hibernate, MySQL, SQL Server y Oracle*. Una vez mapeado, para guardar un objeto en la base de datos simplemente debemos llamar a una función de dicha librería, con la instancia de la clase Java con los valores deseados para guardar el registro.

```
@Override
public void save(AgentGeolocation agentGeolocation) {
    this.dao.save(agentGeolocation);
}
```

*Figura 2.1.2. Implementación del método para guardar un registro concreto en la base de datos*

```
/**
 * Persist the given transient instance, first assigning a generated identifier. (Or
 * using the current value of the identifier property if the <tt>assigned</tt>
 * generator is used.) This operation cascades to associated instances if the
 * association is mapped with {@code cascade="save-update"}
 *
 * @param object a transient instance of a persistent class
 *
 * @return the generated identifier
 */
Serializable save(Object object);
```

*Figura 2.1.3. Declaración del método de la librería de Hibernate para guardar un registro en la base de datos.*

Cabe remarcar que la figura 2.1.2 corresponde a un método para guardar una clase en concreto. Para cada clase Java que queramos guardar en la base de datos se crean en el proyecto diferentes interfaces con sus implementaciones donde se llama, eso sí, al mismo método.

Así pues las diferentes fases del proyecto que se explicarán en los siguientes apartados serán las siguientes (por orden de desarrollo):

- Geolocalización de contactos.
- Geolocalización de agentes automática.
- Geolocalización de CIs.
- Realización de *check-in* y *check-out* para agentes (solo disponible en dispositivos móviles)
- Geolocalización de tickets y problemas.
- Explotación de la información.

1. Véase el apartado *2.2. Geolocalización de contactos* para conocer más sobre la implementación del *composite* para geolocalizar los diferentes elementos de ServiceTonic.

## 2.2. COMPONENTE GEOREGISTER

### 2.2.1. FRONT-END

Se utilizará el elemento *dialog* de la suite de componentes de *PrimeFaces*™ que aparecerá al hacer *click* a un pequeño botón, como se mostrará en los siguientes apartados, situado en un panel utilizado para especificar los datos del ítem que se vaya a georegistrar.

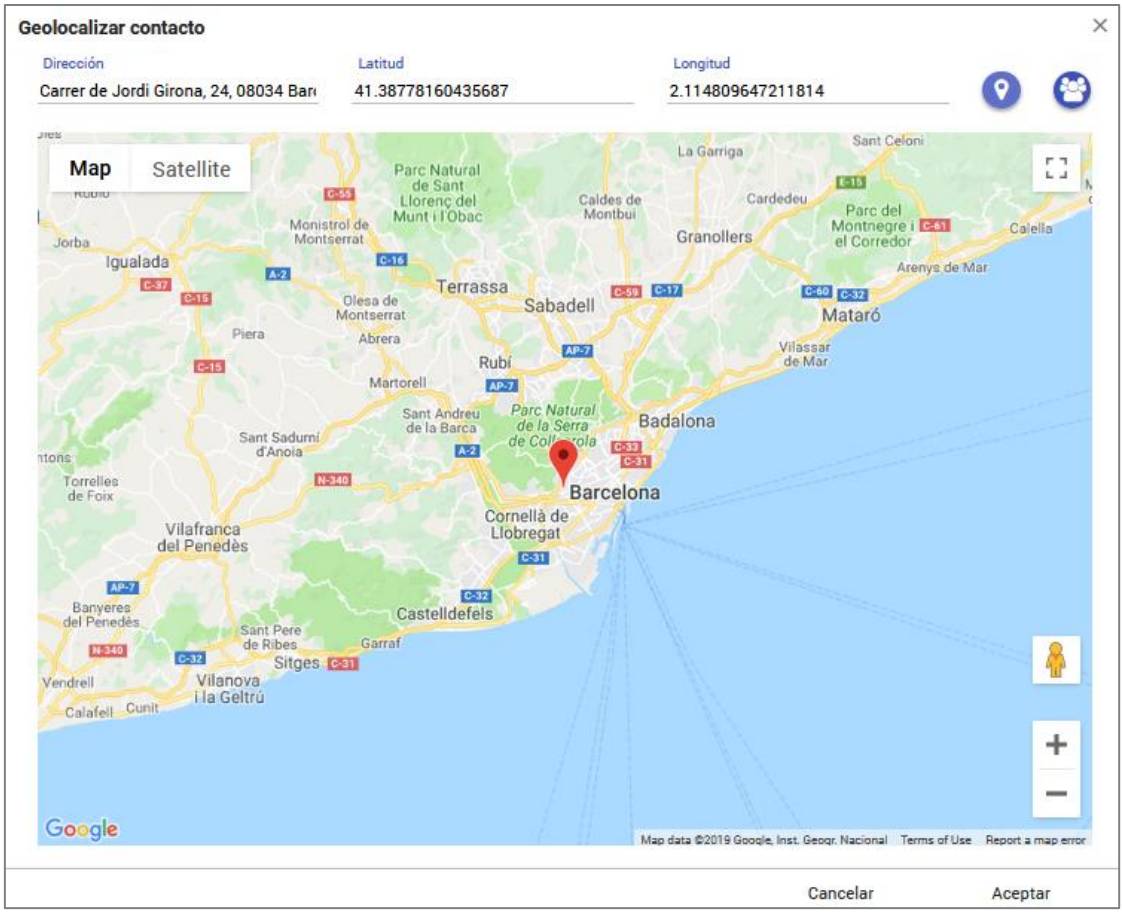


*Figura 2.2.1.1. Diseño final del componente para geolocalizar activos de ServiceTonic.*

El botón abrirá un pop-up con el diálogo anterior donde el usuario tendrá varias formas de especificar la localización:

- Clicando un punto en el mapa. Se rellenarán automáticamente los campos del diálogo con la ubicación del punto seleccionado.
- Clicando el botón situado en la esquina superior derecha del diálogo. Se tomará la ubicación actual del dispositivo y se rellenaran automáticamente los campos del diálogo.
- Introduciendo manualmente la dirección en el campo de dirección. Una vez empiece a introducir la dirección, un servicio de sugerencia de direcciones de Google se mostrará para sugerir direcciones a partir de lo introducido.

Al hacer *click* en el botón de aceptar del diálogo se incluirán los valores indicados en el mismo se traspasaran al panel de registro del ítem.

El botón nos mostrará los agentes disponibles en el mapa para facilitar la búsqueda del agente más cercano del contacto a geolocalizar.

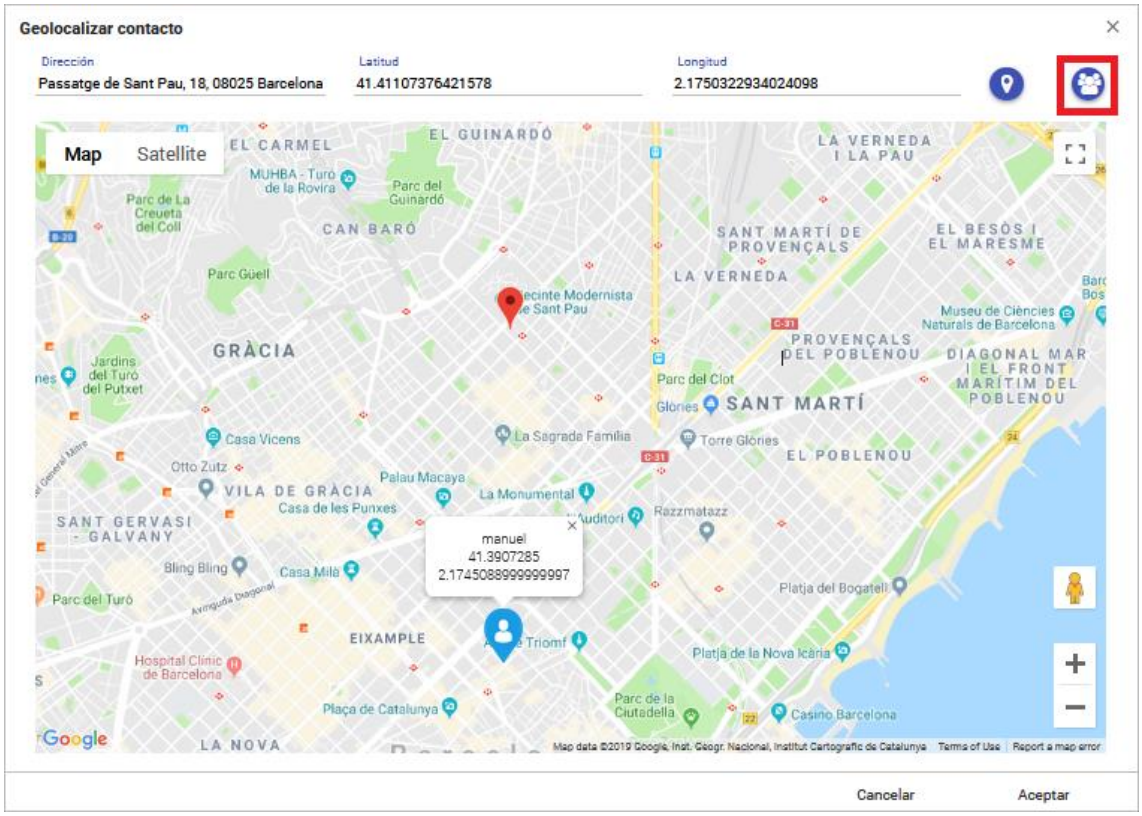


*Figura 2.2.1.2. Funcionalidad de agentes cercanos a la geolocalización seleccionada.*

El hecho de que este diálogo aparezca solamente cuando hacemos *click* en un botón facilita bastante la tarea a la empresa de ocultar este botón a quién no tenga la mejora del servicio de localización incluido en su licencia. Esto mejora la modularidad del producto y del servicio.

Los parámetros modificables del componente serán los siguientes:

- *[address | latitude | longitude] Label*: Etiquetas de los campos de la dirección, latitud y longitud.
- *[address | latitude | longitude] Value*: Valores de los campos de la dirección, latitud y longitud. Estos valores van ligados a las variables declaradas en la clase *ManagedBean* correspondiente.
- *[address | latitude | longitude] Attribute*: Valores de los campos de la dirección, latitud y longitud que serán pasados como atributos a las rutinas del controlador.
- *[cancel | accept] ButtonText*: Texto de los botones de cancelar y aceptar.
- *acceptButtonAction*: Se indica la rutina que se ejecutará al hacer *click* al botón de aceptar.
- *toUpdate*: Se declaran los elementos que deben ser actualizados al final de la ejecución del diálogo, en caso de que hayan cambiado de valor.
- *onPointSelectListener*: Se indica la rutina que se ejecutará al interactuar con el mapa.

```
<!-- Fila 1: Mapa -->
<p:row>
    <p:column colspan="4">
        <p:gmap id="gmap"  widgetVar="wvGmap" center="#{georegisterManagedBean.centerMap}" zoom="10" type="roadmap" styleClass="gmapSize"
                model="#{georegisterManagedBean.simpleModel}">
            <p:ajax event="pointSelect" listener="#{georegisterController.onPointSelect}" update="gmap latitude longitude address"/>
            <p:ajax event="overlaySelect" listener="#{georegisterController.onMarkerSelect}"/>
            <p:gmapInfoWindow id="infoWindow">
                <p:outputPanel style="text-align: center; display: block; margin: auto">
                        <h:outputText value="#{georegisterManagedBean.marker.title}" />
                        <br/>
                        <h:outputText value="#{georegisterManagedBean.marker.data.latitude}" />
                        <br/>
                        <h:outputText value="#{georegisterManagedBean.marker.data.longitude}" />
                </p:outputPanel>
            </p:gmapInfoWindow>

        </p:gmap>
    </p:column>
</p:row>
```

*Figura 2.2.1.3. Fragmento de código de la parte front-end de la incrustación del mapa de Google Maps.*

### 2.2.2. BACK-END

En este apartado se hará una relación de las tres formas de interactuar con el mapa mencionadas en el apartado anterior.

| *Front-end* | *Back-end* |
|---|---|
| *Click* en un punto en el mapa. | Rutina 'onPointSelect' (figura 2.2.2.1) |
| *Click* en el botón | *Script* 'findMeComposite' (figura ) |
| *Click en el botón* | Rutina ' |
| Introducción manual de los datos | - |

En primer lugar, al hacer *click* en un punto del mapa se realiza una llamada Ajax[1] (figura 2.2.2.1) que a partir del evento Ajax registrado toma las coordenadas del punto del mapa y llama a un método de la API de Google Maps para obtener la dirección seleccionada. Una vez obtenido los tres datos se insertan en los campos del componente.

```
public void onPointSelect(PointSelectEvent event) {
    debug("onPointSelect::Inicio");
    //markersViewBean.init();
    LatLng latlng = event.getLatLng();

    //Obtenemos la dirección a partir de las coordenadas
    try {
        JSONObject json = readJsonFromUrl("https://maps.googleapis.com/maps/api/geocode/json?latlng=" +
                latlng.getLat() + "," + latlng.getLng() + "&key=" + this.agentGlobalManagedBean.getMapsAPIKey());
        debug("onPointSelect::URL: " + "https://maps.googleapis.com/maps/api/geocode/json?latlng=" + latlng.getLat()
            + "," + latlng.getLng() + "&key=" + this.agentGlobalManagedBean.getMapsAPIKey());
        this.georegisterManagedBean.setAddress(json.getJSONArray("results").getJSONObject(0).getString("formatted_address"));
    } catch (JSONException | IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

*Figura 2.2.2.1. Fragmento de código de la rutina onPointSelect.*

Por otro lado, si decidimos por obtener la geolocalización del contacto mediante nuestra ubicación actual haciendo *click* en el botón dentro del componente. Se hará una llamada a un método JavaScript (figura 2.2.2.2) para obtener las coordenadas a través de la API de Google Maps.

```
function findMeComposite(addressID, longitudeID, latitudeID) {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(function(position) {
            var latitude = position.coords.latitude;
            var longitude = position.coords.longitude;
            var accuracy = position.coords.accuracy;
            var coords = new google.maps.LatLng(latitude, longitude);
            var latlng = new google.maps.LatLng(latitude, longitude);
            var geocoder = new google.maps.Geocoder();

            geocoder.geocode({ "latLng": latlng }, function (results, status) {
                if (status !== google.maps.GeocoderStatus.OK)
                    alert(status);
                else {
                    setTimeout(function(){
                            document.getElementById(latitudeID).value = latitude;
                            document.getElementById(longitudeID).value = longitude;
                            document.getElementById(addressID).value = results[0].formatted_address;
                            changeInputTextClass();

                    }, 500);
                    onFindMeClickCommand([{name: "lat", value: latitude}, {name: "lng", value: longitude}]);
                }
            })
```

*Figura 2.2.2.2. Fragmento de código del método JavaScript findMeComposite.*

El siguiente botón, el de mostrar los agentes activos del servicio llama a un método (figura 2.2.2.3) el cual muestra la última posición registrada de los agentes almacenados en la base de datos.

```
List<AgentGeolocation> agents = this.geolocationService.getAgentsByProject(sessionManagedBean.getIdProject());
List<Marker> agentMarkers = new ArrayList<>();
for (AgentGeolocation ag : agents) {
    debug("toggleAgents::agent: " + ag.getIdAgentGeolocation() + " " + ag.getLatitude() + " " +
          ag.getLongitude() + " " + ag.getAgent().getUsername());
    if (ag.getLatitude() != null && ag.getLongitude() != null) {
        LatLng latlng = new LatLng(Double.parseDouble(ag.getLatitude()), Double.parseDouble(ag.getLongitude()));
        Marker marker = new Marker(latlng, ag.getAgent().getUsername().getUsername(), ag,
                FacesContext.getCurrentInstance().getExternalContext().getRequestContextPath() + "/images/maps/Agent-azul.png");
        agentMarkers.add(marker);
        this.georegisterManagedBean.getSimpleModel().addOverlay(marker);
    }
}
this.georegisterManagedBean.setAgentPositions(agentMarkers);
```

*Figura 2.2.2.3. Fragmento de código del método toggleAgents.*

Por último, al hacer *click* en el botón de aceptar, a diferencia de los demás botones que accionan métodos del propio componente, este acciona una rutina especificada por atributos del componente. El motivo es que como existen comportamientos distintos para cada ítem al accionar el botón, las diferentes páginas (la de edición de *tickets,* la de problemas, la de edición de CIS, etc.) declaran su propia rutina que se ejecutará al accionar el botón.

## 2.3. GEOLOCALIZACIÓN DE CONTACTOS

### 2.3.1. DESCRIPCIÓN GENERAL

El objetivo de la funcionalidad es la visualización de información geográfica de los contactos en mapas. La funcionalidad estará accesible en la pantalla de edición de tickets y problemas y en la pantalla de mantenimiento de contactos a través de un botón que estará en el panel de edición de datos del contacto asociado al ticket y/o problema. Éste, abrirá un *popup* que mostrará el componente descrito a continuación para seleccionar la geolocalización del contacto.

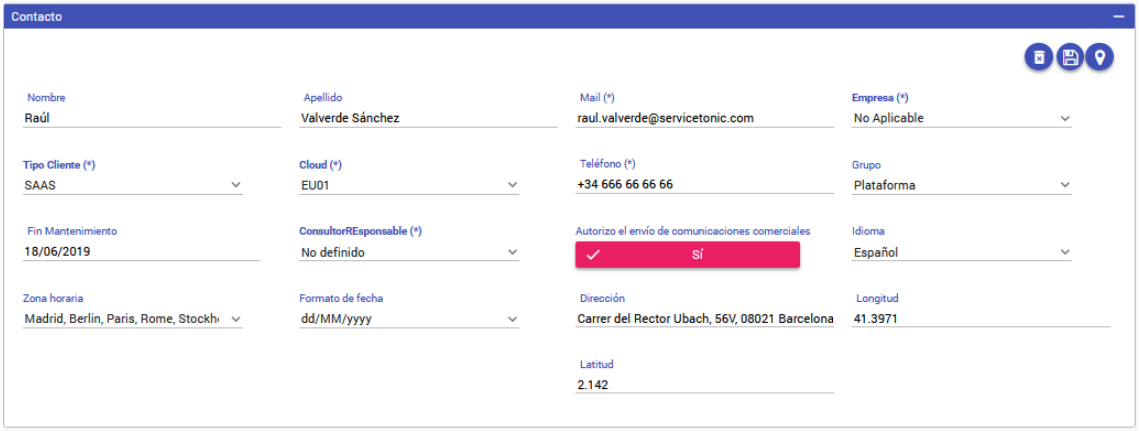


*Figura 2.2.1.1. Panel de datos del contacto asociado al ticket o problema.*

El objetivo principal de esta funcionalidad de geolocalización de contactos no es la de visualizar todos los contactos en un mapa sino la de dotar de geolocalización a un contacto en concreto. Para visualizar todos los contactos del servicio véase el apartado *2.7. Explotación de la información*.

### 2.3.2. MODIFICACIONES EN LA BASE DE DATOS

No se añadirá ninguna tabla nueva, se utilizarán y modificarán las tablas ya existentes en el modelo de la base de datos de la empresa.

Se modificarán 3 tablas del modelo de la base de datos:

- *contact*: Se añadirán los atributos ADDRESS, LATITUDE y LONGITUDE.
- *prj_tplt_field*: Se añadirán los atributos ADDRESS, LATITUDE y LONGITUDE como campos de sistema.
- *prj_tplt_tft*: Se añadirán las traducciones de los atributos utilizados para todos los idiomas en los que está disponible la aplicación.

Como se ha venido comentando en capítulos anteriores, teniendo en cuenta que estamos realizando una modificación en el modelo de base de datos de varias instalaciones del producto de ServiceTonic no podemos pedir a los clientes que modifiquen ellos mismos las bases de datos de sus instalaciones del producto. La solución a este problema es realizar un script que añada automáticamente estas modificaciones a la hora de actualizar el producto con la nueva versión.

### 2.3.3. FUNCIONAMIENTO

Cómo se ha comentado antes hay dos formas de registrar un contacto en el servicio de ServiceTonic:

1. **Creación o edición de tickets**:
   1.1. Hacemos *click* en el apartado de *Service Desk*, en el subapartado de *Todos* (figura 2.3.3.1). Se mostrarán todos los *tickets* del servicio.
   1.2. Una vez se muestren la lista de *tickets* del servicio, hacemos *click* en el aquél donde el contacto que queramos registrar esté implicado.
   1.3. Nos iremos al apartado de *Contacto* (figura 2.2.1.1) y haremos click en el botón para abrir el componente de georegistro, una vez hayamos introducido los demás datos del contacto y seleccionamos la geoposición del contacto mediante una de las formas descritas en el apartado *2.2. Componente Georegister*.
   1.4. Se hace *click* en el botón situado al lado del botón del paso anterior y se guardaran los datos del contacto junto a los datos de geolocalización.

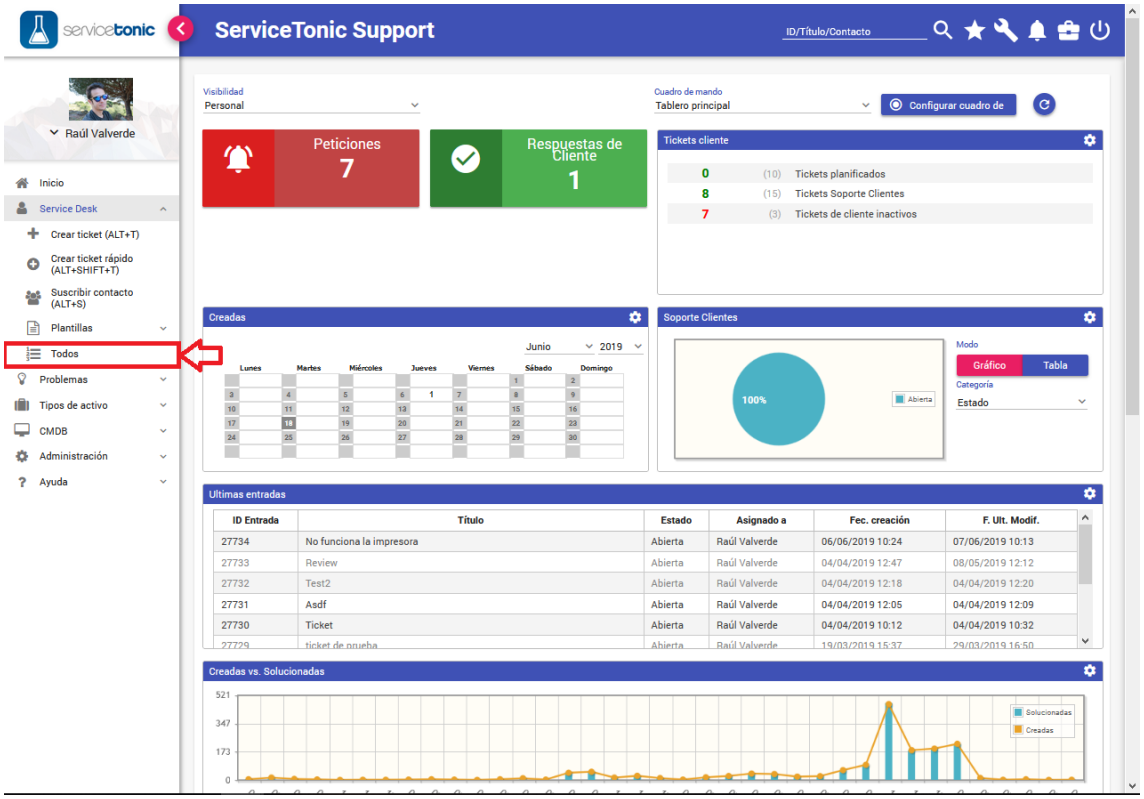


*Figura 2.3.3.1. Paso número 1 para registrar un contacto mediante la edición de tickets.*

2. **Mantenimiento de contactos**:
   2.1. Hacemos *click* en el botón y se nos desplegará una lista de herramientas. Seleccionamos la herramienta de *Contactos* (figura 2.3.3.2).
   2.2. Nos aparecerá la página de mantenimiento de contactos. Podemos hacer *click* en el botón para añadir un contacto nuevo o editar cualquier contacto de la lista
   2.3. En la siguiente página nos aparecerán los datos del contacto y podremos editar de la misma forma que en el apartado anterior *Creación o edición de tickets*.

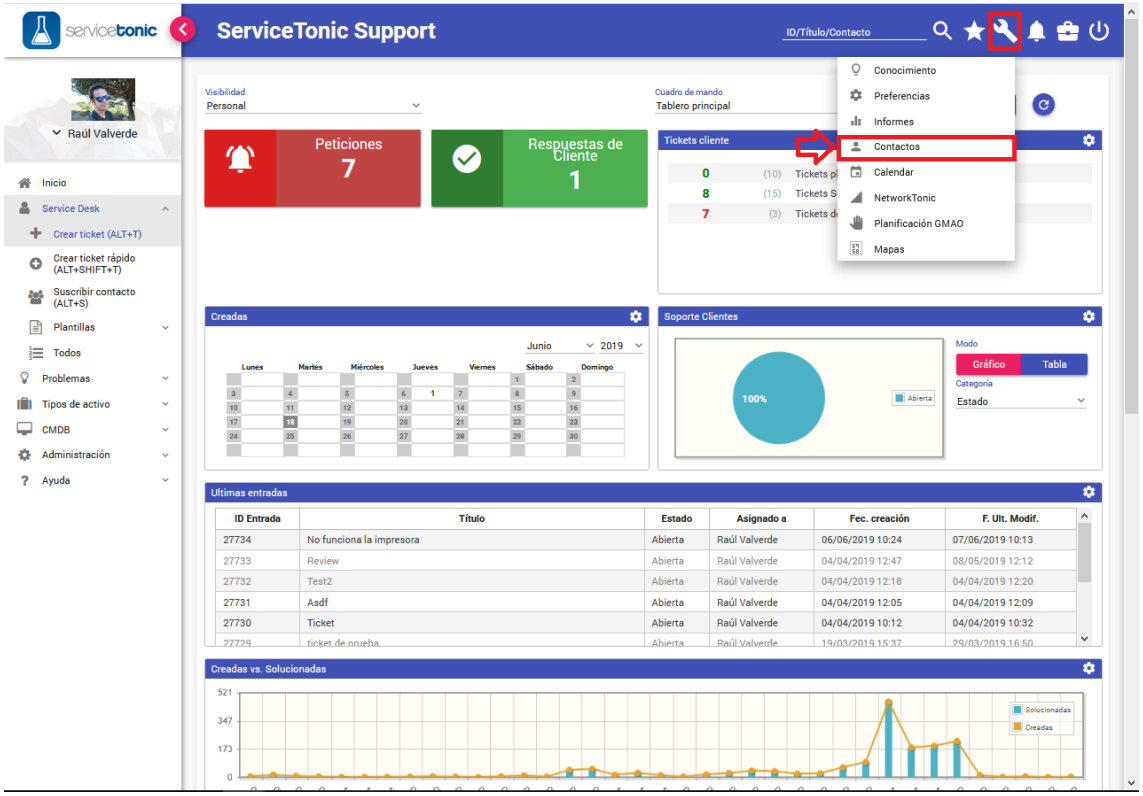


*Figura 2.3.3.2. Paso número 1 para registrar un contacto mediante el mantenimiento de contactos.*

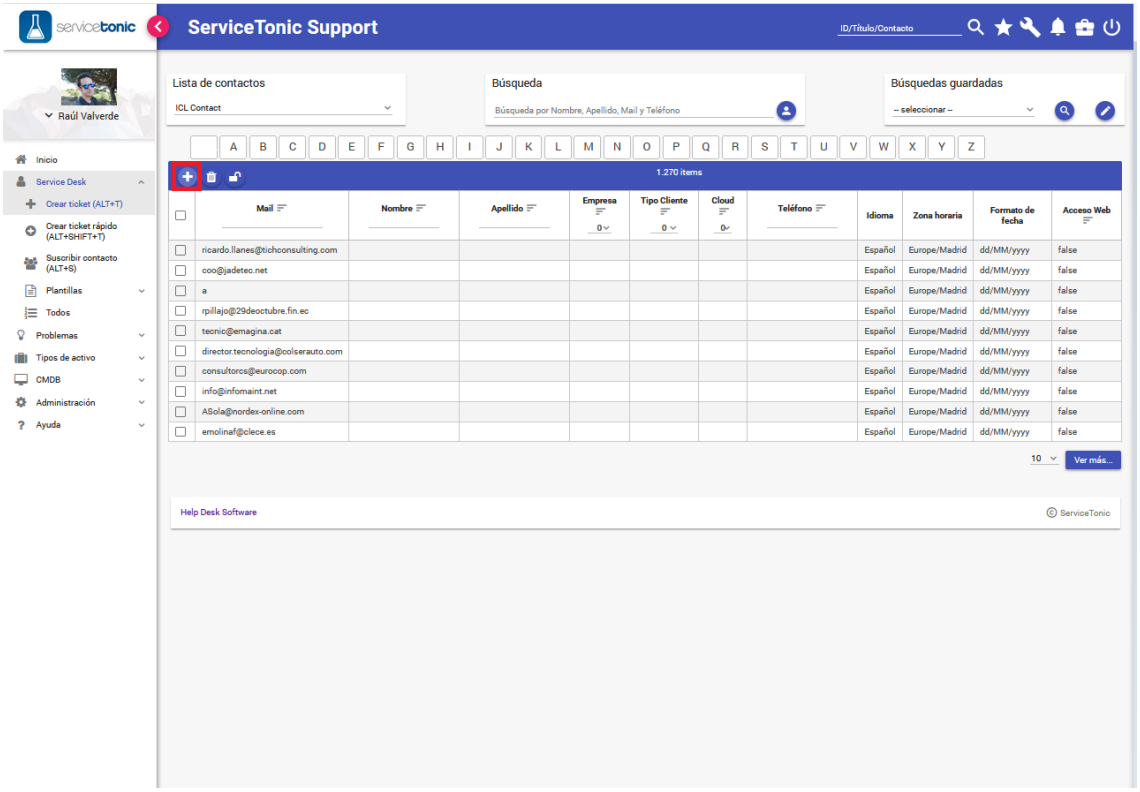


*Figura 2.3.3.3. Paso número 2 para registrar un contacto mediante el mantenimiento de contactos.*

## 2.4. GEOLOCALIZACIÓN DE AGENTES AUTOMÁTICA

La idea en un principio es crear un pequeño sistema de geolocalización automática en segundo plano para dispositivos móviles. El agente podrá saber en cualquier momento si está siendo geolocalizado o no, pudiendo configurar el rango horario en el que podía serlo.

Uno de los objetivos de esta mejora, por otro lado, es la posibilidad de trazar una ruta de un agente que tuviera que desplazarse por diferentes clientes resolviendo problemas. El supervisor tendrá la capacidad de asignarle diferentes *tickets* o problemas según la cercanía de estos al agente.

Se harán uso de las notificaciones *push* a través de *Service Workers*[1], para la comunicación de la geolocalización entre agente, supervisor y cliente.

*Por motivos que se especifican en el apartado 3.1. Conclusiones, problemas y obstáculos, esta mejora finalmente no fue desarrollada.*

1. Un *Service Worker* es un *script* que el navegador ejecuta en Segundo plano, separadamente de la página web, haciendo posible, funcionalidades que no necesitan una página web o la interacción del usuario.

## 1.1. GEOLOCALIZACIÓN DE CIS

### 1.1.1. DESCRIPCIÓN GENERAL

El objetivo de la funcionalidad es bastante similar a la descrita en el apartado *2.3. Geolocalización de contactos*: la visualización de información geográfica de los CIs en mapas. La funcionalidad estará accesible en la pantalla de creación y edición de CIs a través de un botón que estará en el panel de geolocalización del CI. Éste, abrirá un *popup* que mostrará el componente descrito a continuación para seleccionar la geolocalización del CI.

Comentar que está funcionalidad es sólo para dotar a un CI de geolocalización y modificar ésta. Las diferentes geolocalizaciones de todos los activos se podrán visualizar en la herramienta de explotación de la información mostrada en el apartado *2.8. Explotación de la información.*

### 1.1.2. MODIFICACIONES EN LA BASE DE DATOS

No se añadirá ninguna tabla nueva, se utilizarán y modificarán las tablas ya existentes en el modelo de la base de datos de la empresa.

Se modificarán 3 tablas del modelo de la base de datos:

- *configuration_item*: Se añadirán los atributos ST_ADDRESS, ST_LATITUDE y ST_LONGITUDE para geolocalizar el CI.
- *prj_tplt_container*: Se añadirá la pestaña de geolocalización en la página de edición y creación de CIs.
- *prj_tplt_tct*: Se añadirán las traducciones de los atributos utilizados para todos los idiomas en los que está disponible la aplicación.

### 1.1.3. FUNCIONAMIENTO

El funcionamiento es muy similar al de geolocalizar contactos, explicado en el apartado *2.3. Geolocalización de contactos*. La funcionalidad está disponible solamente, en este caso, desde la página de creación y edición de CIS.

1. **Creación y edición de CIs**:
   1.1. Nos dirigimos a la pestaña de CMDB y seguidamente hacemos *click* a *Crear CI* si queremos crear un CI nuevo que queremos geolocalizar o *All* si queremos ver la lista de CIs del servicio y seleccionar uno (figura 2.5.3.1).
   1.2. Si hemos hecho *click* en *All* nos aparecerá una lista con todos los CIs del servicio, haremos *click* entonces a aquel CI que queramos geolocalizar (figura 2.5.3.2).
   1.3. Una vez estemos en la página de creación de CIs o edición de CIs, que tienen exactamente el mismo aspecto nos dirigiremos a la pestaña de *Geolocalización* y haremos *click* en el botón.
   1.4. Nos aparecerá el componente de georegistro e introduciremos los datos a través de una de las formas explicadas en el apartado *2.2. Componente Georegister*.
   1.5. Una vez hayamos introducido todos los datos haremos *click* al botón de *Guardar*.

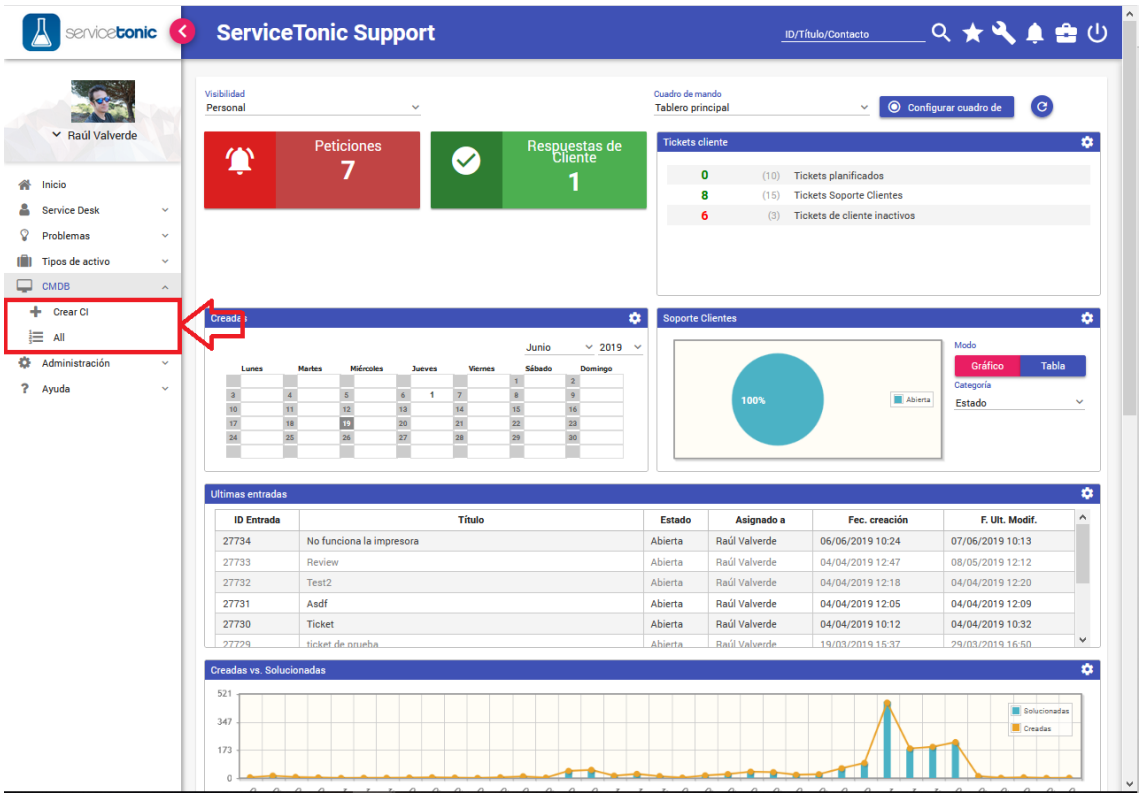


*Figura 2.5.3.1. Primer paso para geolocalizar un CI.*

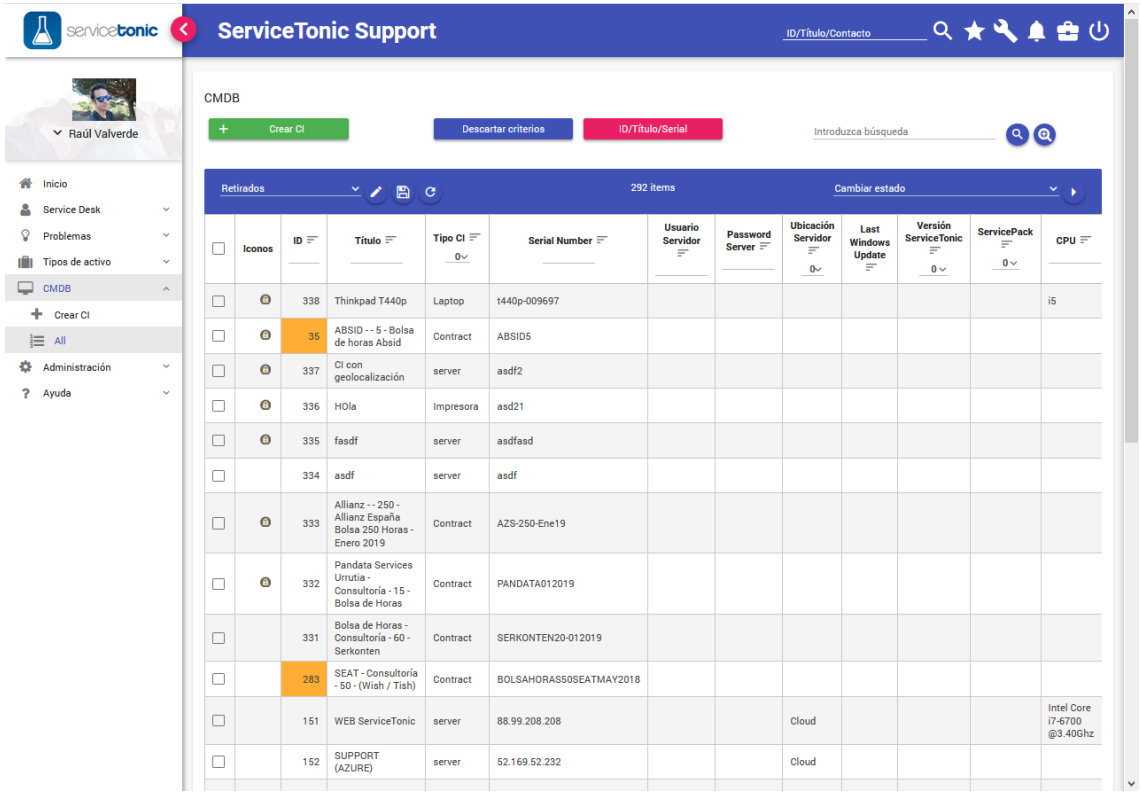


*Figura 2.5.3.2. Segundo paso para geolocalizar un CI. Lista de CIs del servicio.*

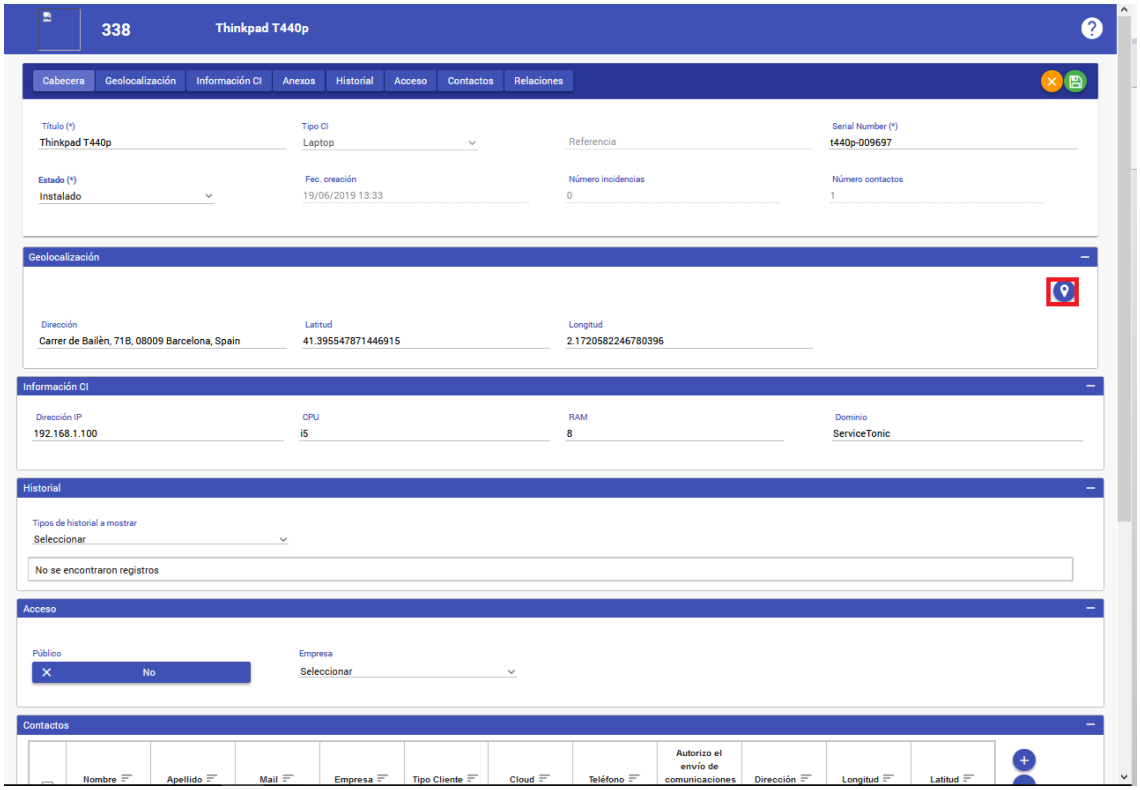


*Figura 2.5.3.3. Tercer paso para geolocalizar un CI. Página de creación/edición de un CI.*

## 1.2. CHECK-IN – CHECK-OUT

### 2.6.1. DESCRIPCIÓN GENERAL Y FUNCIONAMIENTO INTERNO

El objetivo de esta mejora es implementar un pequeño sistema de control a través de *check-in* y *check-out* disponible para aquellos agentes que se desplacen a otros lugares para resolver cualquier incidencia que tengan relacionada a ellos.

Es por el motivo del párrafo anterior que la mejora sólo estará accesible a través de la versión de ServiceTonic para *smartphones*. Con esta afirmación me refiero que realizar los *check-ins* y *check-outs* sólo se podrá realizar en dispositivos móviles, la visualización de estos datos estará disponible en cualquier dispositivo, siempre y cuando disponga de los permisos necesarios.

El funcionamiento interno de la mejora es la siguiente:

- Al hacer *check-in*: lo hará el agente cuando llega al cliente, donde ha surgido la incidencia.
    - La posición georegistrada en el momento del *check-in* va asociada al agente y al ticket. Cuando se grabe el *check-out* además irá asociada a la actuación[1].
    - Si el *ticket* tenía un *check-in* sin ningún *check*-out se sobrescribirá la información del *check-in* anterior almacenada en el *ticket*.
    - En el momento del *check-in* se almacenará la hora, la posición y el agente que ha realizado el *check-in*.

- Al hacer *check-out* (se graba una actuación en el *ticket*):
    - Si el ticket tenía un *check-in* pendiente:
        - Si el agente es el mismo:
            - Se grabará en la actuación la posición y la hora del *check-in,* almacenadas en el ticket.
            - Se grabará en la actuación la posición y la hora del *check-out*.
            - Como tiempo de actuación grabará la diferencia los tiempos del *check-in* y *check-out* a no ser que introduzca manualmente un tiempo. En este último caso grabará la información manual.
        - Si el agente es diferente:
            - Se eliminará la información del *check-in* previo, realizado por un agente diferente, del ticket
            - Se grabará como *check-in* la actuación la posición y la hora actuales
            - Se grabará en la actuación la posición y la hora del *check-out*.
            - Como tiempo de actuación grabará la que se introduzca manualmente.

1. Una actuación, es cualquier acción que se haya hecho sobre un *ticket o problema*. Esta puede ser la solución final al problema o simplemente una aclaración, una duda, un problema con la posible solución, etc.

- Si el ticket NO estaba en *check-in*:
  - Se grabará como check-in la actuación la posición y la hora actuales.
  - Se grabará en la actuación la posición y la hora del *check-out*.
  - Como tiempo de actuación grabará la que se introduzca manualmente.

### 2.6.2. MODIFICACIONES EN LA BASE DE DATOS

Se creará una nueva tabla, con el objetivo de guardar las posiciones y las fechas de los *check-ins* y *check-outs* de los agentes:

- *Agent_geolocation*: Los atributos que almacenará esta tabla serán los siguientes:
  - ID_AGENT_GEOLOCATION
  - ID_AGENT
  - TYPE
  - CREATION_DATE
  - LATITUDE
  - LONGITUDE

Se modificarán 2 tablas del modelo de la base de datos:

- *Incident*: Se añadirán los atributos CHECKIN_AGENT, CHECKIN_DATE, CHECKIN_LATITUDE, CHECKIN_LONGITUDE, CHECKOUT_AGENT, CHECKOUT_DATE, CHECKOUT_LATITUDE, CHECKOUT_LONGITUDE, ST_LATITUDE, ST-LONGITUDE y ST_ADDRESS, para almacenar la información relativa al agente y a los *check-ins* y *check-outs* de éste en la información del *ticket*.
- *Incident_action*: Se añadirán los atributos CHECKIN_DATE, CHECKIN_LATITUDE, CHECKIN_LONGITUDE, CHECKOUT_DATE, CHECKOUT_LATITUDE, CHECKOUT_LONGITUDE, para almacenar la información relativa al agente y a los *check-ins* y *check-outs* de éste en la información de cada actuación realizada sobre un mismo ticket.

### 2.6.3. FUNCIONAMIENTO (USUARIO)

En este apartado se explicará el funcionamiento de la mejora de cara al usuario, comentando paso a paso el procedimiento a seguir.

Supongamos que un agente de la empresa ServiceTonic ha de desplazarse a *Campus Nord* porque tiene que resolver un problema relacionado con la instalación del *software* de ServiceTonic. Los pasos que debería seguir serían los siguientes:

1. Realización del *check-in*:
   1.1. Al llegar a *Campus Nord* haría *login* en la aplicación móvil de ServiceTonic (figura 2.6.3.1).
   1.2. Una vez hecho *login*, le aparecerían los *tickets* del servicio que tiene disponibles. Una vez localizado el *ticket* en cuestión que ha de resolver haría un *swipe* a la izquierda y le aparecería una serie de botones (figuras 2.6.3.2 y 2.6.3.3).

1.3. Para realizar el *check-in* haría *click* en el botón  . En el caso de que existiese un *check-in* previo se le mostraría una confirmación para sobrescribirlo (figura 2.6.3.4). Una vez realizado el *check-in* se le mostraría un aviso de que el *check-in* se ha realizado correctamente en el caso de que todo haya ido bien (figura 2.6.3.5).

2. Realización del ***check-out***:
   2.1. Una vez el agente haya terminado el trabajo necesario en el cliente realiza el *check-out* registrando una actuación sobre el *ticket* en cuestión. Para ello, ha de dirigirse a la pantalla del *ticket* {figura 2.6.3.6).
      2.1.1. Si decide no registrar ningún comentario en la actuación, se grabará con el comentario por defecto, el cual es *Check-out*.
      2.1.2. Si decide no registrar ningún tiempo de actuación, se grabará la diferencia entre el tiempo de *check-in* y el *check-out* en el tiempo de actuación.
   2.2. Una vez el agente haya introducido los datos necesarios hará *click* en el botón de *Guardar Ticket*  . La aplicación saldrá a la pantalla principal la cuál es la lista de *tickets* disponibles en el servicio.

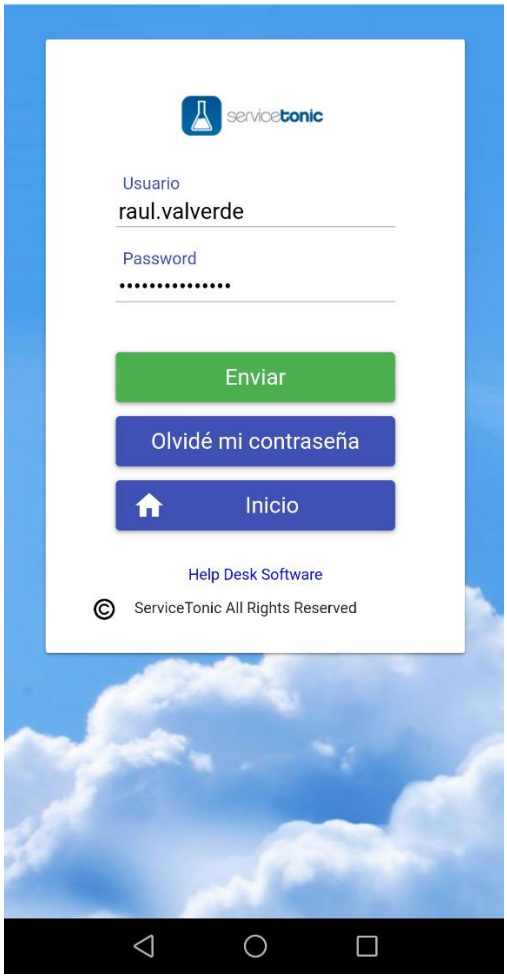


*Figura 2.6.3.1. Pantalla de login de la aplicación móvil de ServiceTonic*

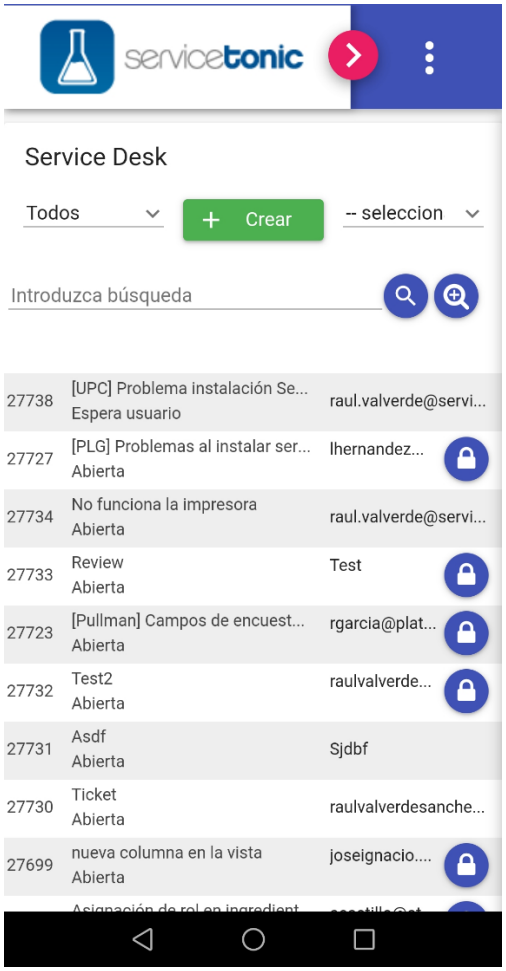


*Figura 2.6.3.2. Lista de tickets disponibles del servicio.*

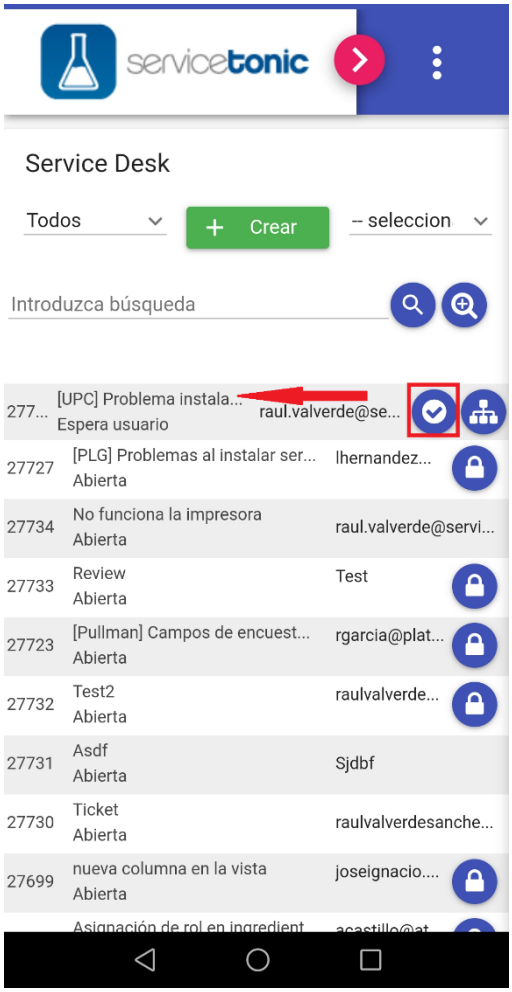


*Figura 2.6.3.3. Botón de check-in de un ticket del servicio.*

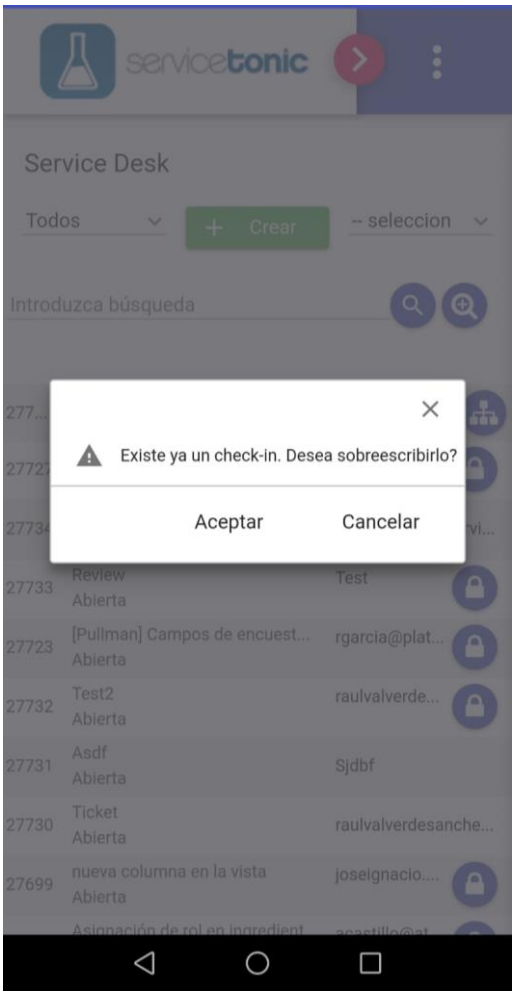


*Figura 2.6.3.4. Aviso de sobrescribir un ticket del servicio.*

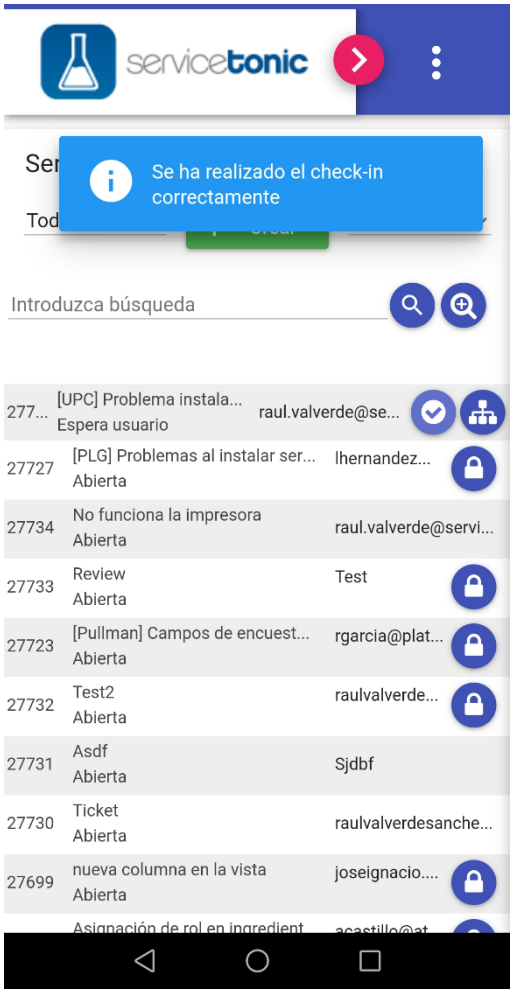


*Figura 2.6.3.5. Realización correcta del check-in de un ticket del servicio.*


*Figura 2.6.3.6. Grabación de la actuación del check-out de un ticket del servicio.*

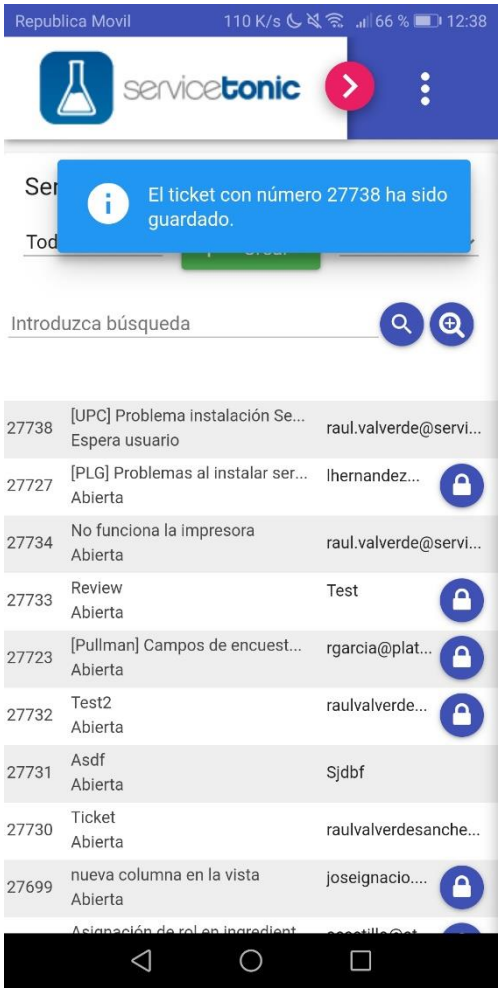


*Figura 2.6.3.7. Grabación correcta de la actuación del check-out de un ticket del servicio.*

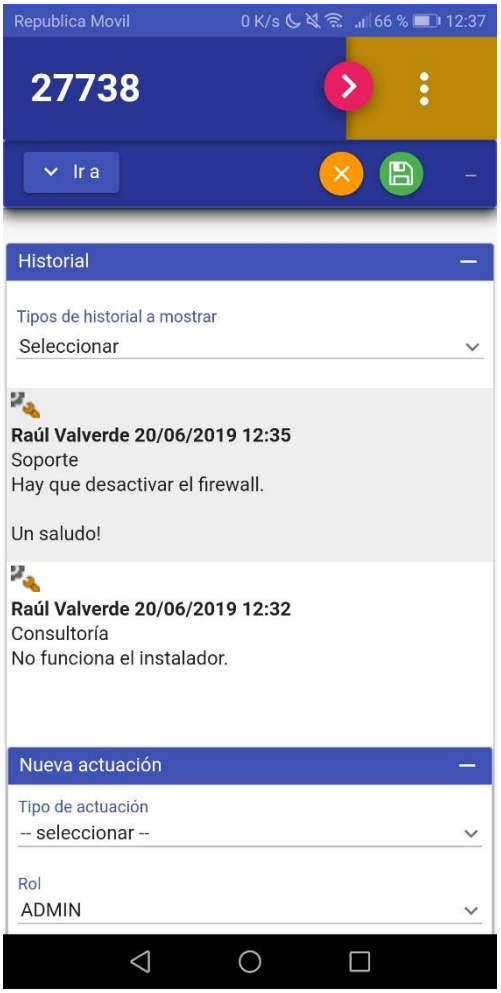


*Figura 2.6.3.8. Historial de actuaciones de un ticket del servicio.*

## 2.7. GEOLOCALIZACIÓN DE TICKETS

### 2.7.1. DESCRIPCIÓN GENERAL

El objetivo de la mejora es registrar la posición de los tickets.

Se registrarán una de las siguientes posiciones para cada *ticket* que se desee geolocalizar:

- Ubicación actual (sólo en dispositivos móviles).
- Marcar punto en el mapa del componente de georegistro.
- Heredar de uno de los CIs vinculados
- Heredar del contacto

El tipo de posición a registrar por defecto será configurable tanto para movilidad como para escritorio. Se podrá configurar pues, para aquellos *tickets* que no tengan geolocalización, tanto sea para nuevos *tickets* o no, el orden del tipo de geolocalización que se quiere por defecto, las diferentes posibilidades son:

- Si se está creando el *ticket* en dispositivos móviles:
  - Posición actual del dispositivo.
  - Posición del contacto del *ticket*.
  - Posición de uno de los CIs vinculados, si tiene.
- Si se está creando el *ticket* en un dispositivo de escritorio:
  - Posición del contacto del *ticket*.
  - Posición de uno de los CIs vinculados, si tiene.

Se podrá cambiar la posición del ticket durante la vida del mismo. Esta mejora aplica tanto a *tickets* como a problemas.

Se aprovechará la mejora para añadir una nueva funcionalidad al composite de geolocalización. Se incorporará el botón *Mostrar agentes*[1], de forma que nos muestre la última posición registrada de cada agente del servicio.

### 2.7.2. MODIFICACIONES EN LA BASE DE DATOS

Solamente hará falta modificar una tabla debido a que ya se hicieron las modificaciones necesitarías en el apartado 2.6. Check-in check-out. Así pues, las modificaciones necesarias en esta función son las siguientes:

1. El botón de *Mostrar Agentes* se describe en el apartado *2.2. Componente georegister*

Se han añadido las siguientes variables de módulo:

- module.geolocationDesktopOriginDefault. Almacenará qué se ha de tomar por defecto para geolocalizar un ticket. Los valores posibles son las 2 permutaciones posibles de CONTACT y CI. El valor por defecto será NULL.
- module.geolocationMobileOriginDefault. Almacenará qué se ha de tomar por defecto para geolocalizar un ticket. Los valores posibles son las 6 permutaciones posibles de POSITION, CONTACT y CI. El valor por defecto será POSITION- CI-CONTACT.

Las variables de módulo anteriores son las que se mostrarán en la página de configuración de *ServiceDesk*.

### 2.7.3. FUNCIONAMIENTO

De cara al usuario, tendrá un aspecto muy similar a mejoras anteriores. Será un panel llamado *Geolocalización* situado en la pantalla de creación/edición de *tickets*.

El panel de geolocalización tendrá los tres campos de geolocalización, un desplegable con los CIs vinculados a ese *ticket*, un botón que abrirá el componente de georegistro y un nuevo botón

el cuál traspasará la información de la posición del contacto del *ticket* a los campos de la pestaña de geolocalización.

Así pues, el funcionamiento de la mejora es el siguiente:

1. Configuración valores de geolocalización por defecto:
   1.1. Haremos click en la pestaña de *Administración Service Desk* (figura 2.7.3.1).
   1.2. Configuraremos el orden deseado en los desplegables de *Geolocalización por defecto escritorio* y *Geolocalización por defecto movilidad* (figura 2.7.3.2).

2. Establecimiento de la geolocalización de un *ticket* o problema:
   2.1. Entraremos en la página de edición de *tickets* o problemas del *ticket* o problema que deseemos geolocalizar.
   2.2. Iremos a la pestaña de *Geolocalización* (figura 2.7.3.3).
   2.3. Aquí tenemos tres opciones, comentadas anteriormente:
      2.3.1. Geolocalizar el *ticket* o problema haciendo *click* en el botón . Se nos abrirá el componente de georegistro y seleccionaremos la geoposición deseada. El funcionamiento del componente de georegistro se describe en el apartado *2.2. Componente georegister.*
      2.3.2. Haciendo *click* en el botón . Se traspasará la información de la geolocalización del contacto a los campos de geolocalización del *ticket*.
      2.3.3. Seleccionando uno de los CIs vinculados del desplegable. El *ticket* tomará la geolocalización del CI seleccionado.

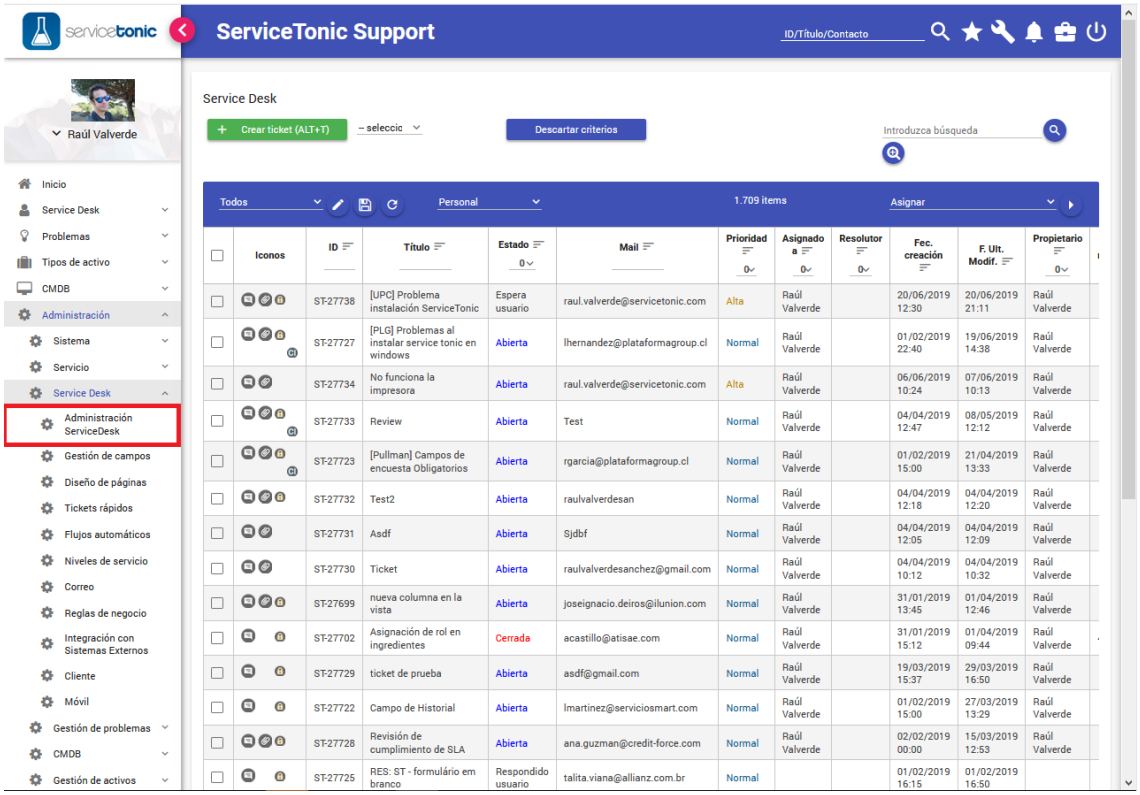


*Figura 2.7.3.1. Apartado de configuración ServiceDesk.*

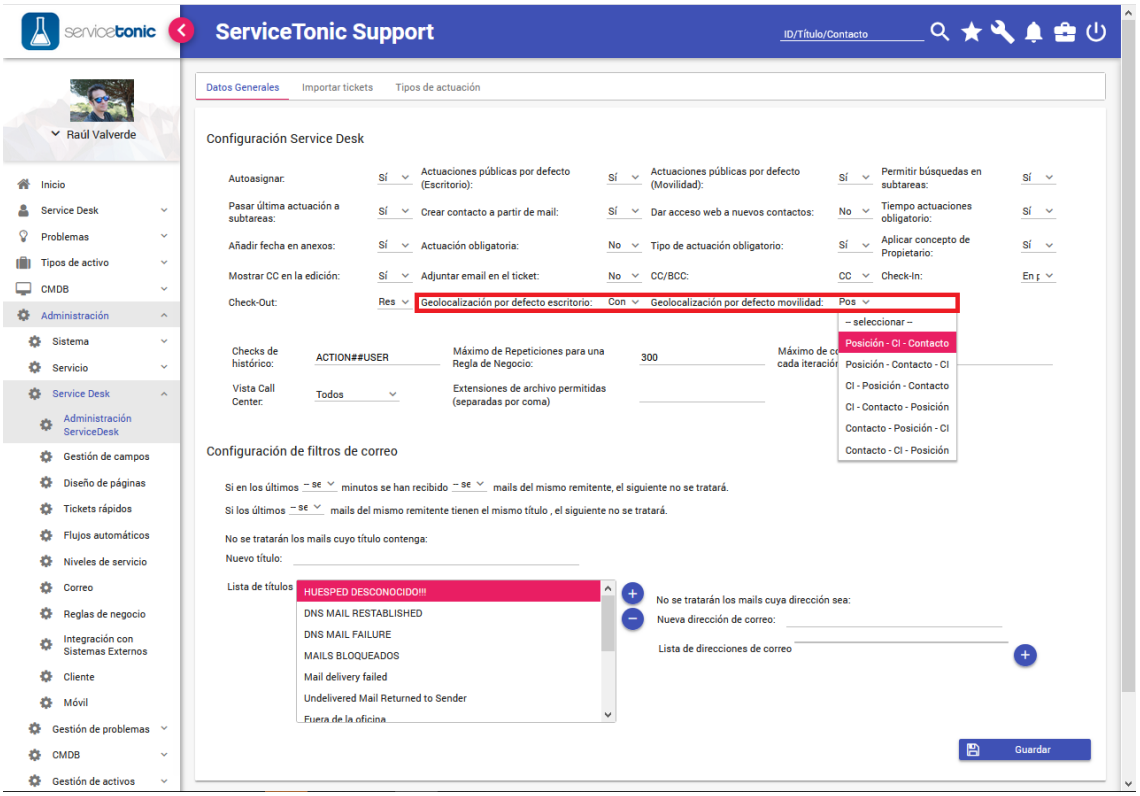


*Figura 2.7.3.2. Apartado de configuración ServiceDesk.*

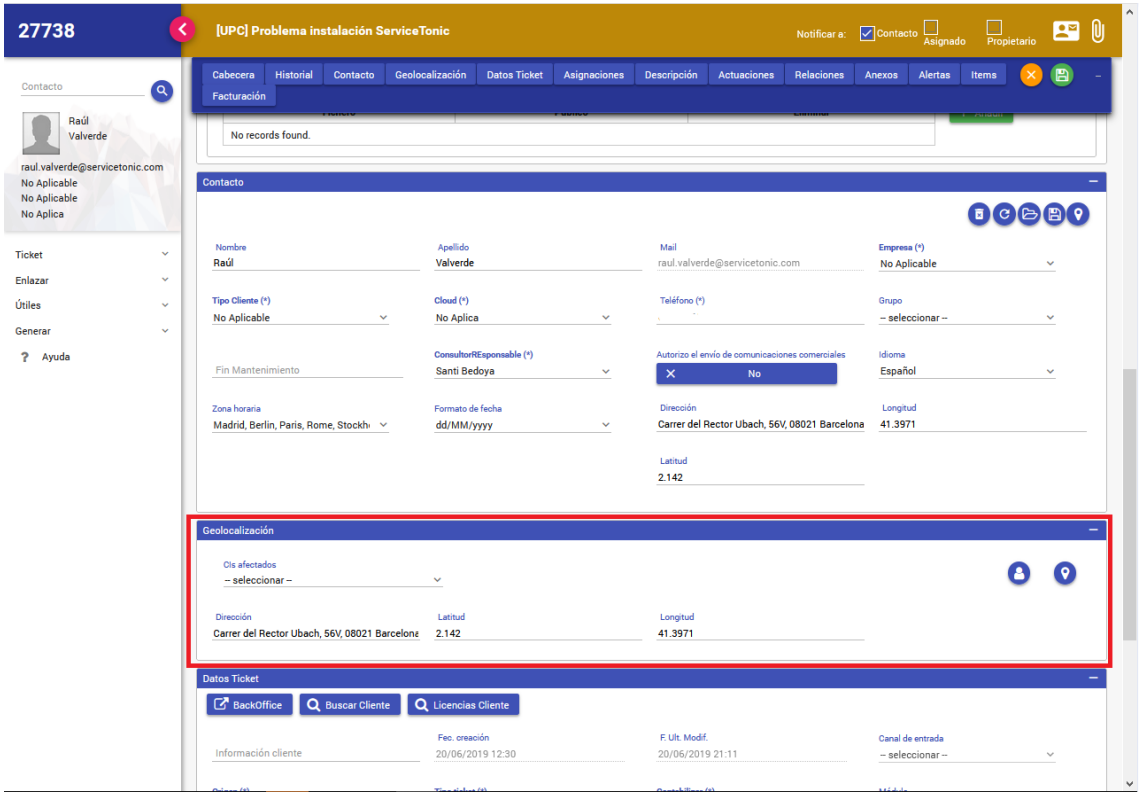


*Figura 2.7.3.3. Pestaña de geolocalización de un ticket.*

## 2.8. EXPLOTACIÓN DE LA INFORMACIÓN

### 2.8.1. DESCRIPCIÓN GENERAL

Ésta será la última funcionalidad del proyecto. El objetivo de ésta es poder explotar la información de geolocalización que hemos ido almacenando en el resto de las mejoras del proyecto. Con ella, podremos visualizar toda la información, ítems, *tickets* geolocalizados en los apartados anteriores en un mismo mapa especificando los parámetros concretos necesarios para su funcionamiento.

En concreto, el sistema es capaz de proporcionarnos la siguiente información:

- Contactos:
  - Parámetros: Se podrá buscar por nombre, apellido, mail o teléfono, o, por otra parte, seleccionando una búsqueda de contacto[1] guardada a partir de un desplegable.
  - Información obtenida: La información que se obtendrá será los marcadores representados en el mapa donde cada marcador representará un contacto. Al hacer *click* en cualquier marcador, que tendrá forma de contacto, se desplegará una tarjeta donde se mostrará el título del contacto y, por otro lado, los cinco primeros campos configurados previamente, por ejemplo, nombre, apellidos, *email*, etc.
  - Formato de la información: Al hacer *click* en cualquier marcador, que tendrá forma de contacto, se desplegará una tarjeta donde se mostrará lo siguiente:
    - El título del contacto, que será el identificador del contacto en el sistema.
    - Los 5 primeros campos del contacto configurados previamente, por ejemplo, nombre, apellidos, *email*, etc.
    - La geolocalización del contacto.
    - Un botón que nos redirigirá a la página para editar el contacto seleccionado.
    - Un botón para visualizar ese punto en *Google Maps*.
- CI:
  - Parámetros: Se podrá buscar por el ID del CI o por su título, además se dispondrá de un desplegable donde podremos seleccionar una vista de CIs de entre las que tengamos configuradas, como se explicará a continuación.
  - Información obtenida: Obtendremos las posiciones de los CIs filtrados por los criterios anteriores en forma de marcadores con una forma genérica en forma de marcador.
  - Formato de la información: Al hacer *click* en cualquier marcador, que tendrá forma de contacto, se desplegará una tarjeta donde se mostrará lo siguiente:
    - El título del CI, que será el tipo del CI (servidor, pantalla, *laptop*, etc.), seguido del número de serie y del título.
    - Los 5 primeros campos del CI configurados previamente, por ejemplo, estado, título, id, serial, etc.
    - La geolocalización del CI.
    - Un botón que nos redirigirá a la página para editar el CI seleccionado.
    - Un botón para visualizar ese punto en *Google Maps*.

1. Una **búsqueda de contactos** es una agrupación de contactos que han sido filtrados por una serie de criterios. Por ejemplo, se puede crear una búsqueda predefinida que sea 'ServiceTonic' donde los contactos que se muestren sean pertenecientes a ServiceTonic.

- Agentes:
  - Parámetros: Los parámetros, en este caso, son un poco más complejos que los anteriores. En primer lugar, podremos configurar el rango de tiempo entre los que deberían reflejar los registros de los agentes. Por otro lado, se dispone de dos desplegables; en el primero, se seleccionará uno o varios equipos de agentes (desarrollo, comercial, *marketing*, o como se tenga configurado). A partir de los equipos seleccionados en el desplegable podremos seleccionar a otro a los agentes disponibles que queremos geolocalizar que pertenecen a los equipos seleccionados.
  - Información obtenida: Obtendremos las posiciones de los agentes filtrados por los criterios anteriores en forma de marcadores con una forma genérica en forma de agente. Cada marcador tendrá un color diferente según si su posición registrada fue un *check-in* o un *check-out*.
  - Formato de la información: Al hacer *click* en cualquier marcador, que tendrá forma de contacto, se desplegará una tarjeta donde se mostrará lo siguiente:
    - El título del agente, que será el usuario del agente.
- Tickets:
  - Parámetros:
    - Rango de tiempo
    - Lista de vistas de tickets
    - Buscador por ID/Titulo/Contacto
  - Información obtenida:
    - Posición de los tickets en el mapa. Mostrar diferentes colores si es asignado, checkin o checkout.
  - Formato de la información: Al hacer *click* en cualquier marcador, que tendrá forma de contacto, se desplegará una tarjeta donde se mostrará lo siguiente:
    - El título del *ticket*, que será el identificador del contacto asociado al *ticket*, seguido del usuario del agente asignado y el título del *ticket*.
    - Los 5 primeros campos del *ticket* configurados previamente, por ejemplo, estado, título, id, fecha creación, etc.
    - La geolocalización del *ticket*.
    - Un botón que nos redirigirá a la página para editar el *ticket* seleccionado.
    - Un botón para visualizar ese punto en *Google Maps*.

Por otro lado, cabe remarcar que el usuario antes de usar esta mejora deberá configurar, como se explicará a continuación, las vistas de tickets y CIs que querrá configurar para poder ser geolocalizadas y mostradas en esta mejora, así como las búsquedas de contacto correspondientes.

### 2.8.2. MODIFICACIONES EN LA BASE DE DATOS

En esta última mejora solamente hará falta modificar algunas tablas de la base de datos con tal de poner en marcha esta mejora:

- *Incident*: Se añadirá la columna PLANNED_DATE. Esta columna permitirá establecer la fecha planificada de resolución de un *ticket* o problema. El atributo será consultado a la hora de filtrar los *tickets* por fecha.
  - *Prj_tplt_field*: Se añadirá el campo de sistema de PLANNED_DATE para integrarlo en el sistema.

  - *Prj_tplt_tft*: Se crearán las traducciones de este campo para todos los idiomas en los que la aplicación está disponible.

- *Guest*: Se añadirá el campo MAPS_API_KEY para administrar la licencia de *Google Maps*.

- *View_definition*: Se añadirá el atributo MAPS_VIEW_ORDER. Este valor se establecerá a la hora de establecer el orden en el que queremos configurar las vistas de *tickets* o CIs para ser geolocalizadas.

- *Contact_search:* Se añadirá el atributo MAPS_VIEW_ORDER. Este valor se establecerá a la hora de establecer el orden en el que queremos configurar las búsquedas de contactos para ser geolocalizadas.

- *Tool*: Se añadirá una entrada al menú de *Herramientas* para hacer la herramienta de *Maps* accesible.

- *Admin_icon*: Se dará de alta la nueva opción de administración.

- *Admin_icon_option*: Se dará de alta la nueva opción de administración.

### 2.8.3. FUNCIONAMIENTO

En primer lugar, como se ha comentado antes, el usuario ha de configurar primeramente las vistas de *tickets* y CIs y las búsquedas de contactos que vaya a querer saber la geolocalización en un futuro. Una vez configuradas, el usuario ya podrá hacer uso de la mejora adecuadamente.

Una vez dicho esto, los pasos a seguir para hacer uso de la mejora son los siguientes:

1. Configuración de las vistas:
   1.1. Nos dirigiremos a la pestaña de *Mapas* en el apartado de *Servicio* dentro de *Administración* (figura 2.8.3.1).
   1.2. Nos aparecerá una pantalla donde podremos configurar por un lado la clave de la API de *Google Maps* y por otro lado tendremos un panel con tres categorías: *Agentes/Tickets, CIs* y *contactos* (figura 2.8.3.2).
   1.3. Para configurar una vista o búsqueda de contactos para que pueda ser mostrada como geolocalizable seleccionaremos las deseadas y las pasaremos al cuadro derecho.
   1.4. Una vez hayamos establecido la configuración deseada haremos *click* en el botón de *Guardar*.

2. Página de explotación de la información:
   2.1. Nos dirigiremos al icono y desplegará una serie de herramientas. Seleccionamos la herramientas de *Mapas* (figura 2.8.3.3).
   2.2. Se nos mostrará entonces, la herramienta de *Mapas*. Haremos *click* en los *checkboxes* de las categorías que quereamos geolocalizar y se habilitarán los campos de las que hayamos seleccionado.

2.3. Seleccionamos las vistas o búsquedas de contactos deseadas y rellenamos el campo de búsqueda si queremos buscar un elemento en concreto de la lista o si no, lo dejamos vacío para que busque todos los elementos de la lista que hayamos seleccionado

2.4. En el apartado de *Tickets*/Agentes seleccionamos una de las dos en caso de que hayamos seleccionado el *checkbox* correspondiente.

2.5. Introducimos el rango de fechas si deseamos filtrar por fecha.

2.6. En caso de que hayamos seleccionado *Tickets* seleccionamos la vista de *tickets* deseada e introducimos la búsqueda si así lo preferimos.

2.7. En caso de que hayamos seleccionado *Agentes* seleccionamos los equipos que queramos seleccionar y seleccionamos los agentes que queramos geolocalizar.

2.8. Una vez hayamos introducido todos los criterios deseados hacemos *click* en cualquier de los botones de búsqueda.

2.9. Nos aparecerán los marcadores en el mapa, individualmente o clusterizados en caso de que esten muy cerca (figura 2.8.3.4).

2.10. En caso de que no haya aparecido el que queramos, en el caso de que no hayamos introducido ningún *token* de búsqueda, podemos hacer *click* en el botón de *Ver más* y se renderizarán más geolocalizaciones que cumplan con los criterios de búsquedas.

2.11. Podemos hacer *click* en cualquiera de ellos y se desplegará una tarjeta con diferentes datos.

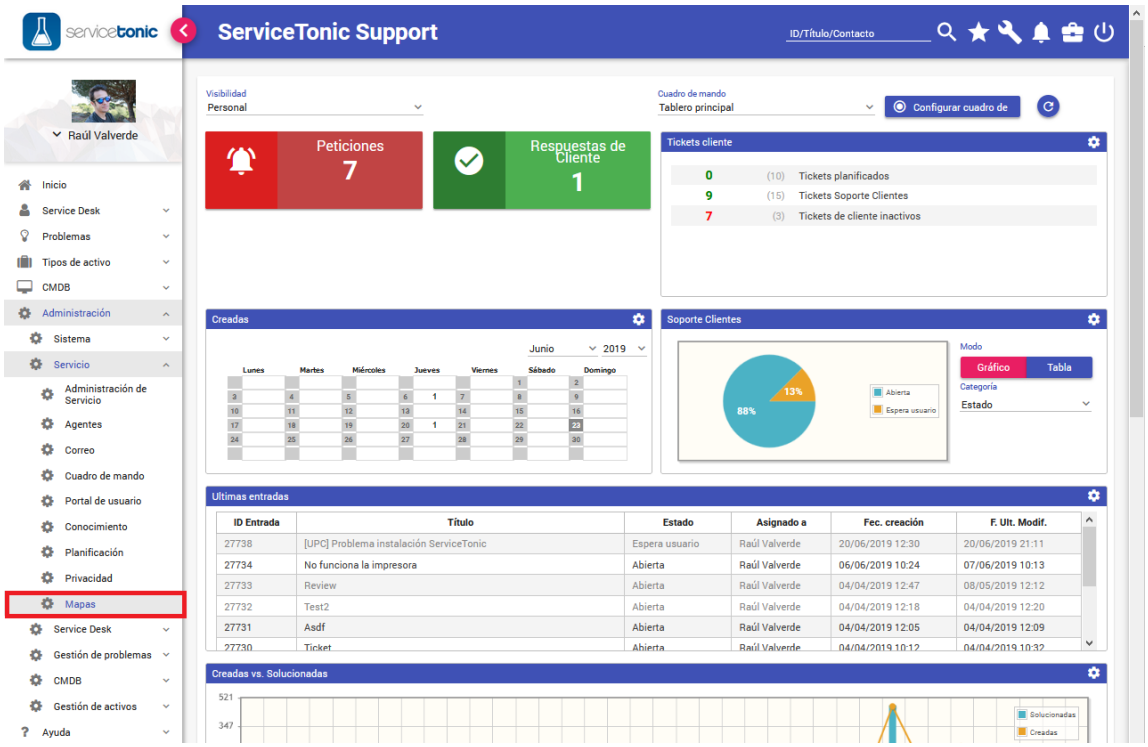


*Figura 2.8.3.1. Pestaña de Mapas. Configuración de las vistas y búsquedas de contactos geolocalizables.*

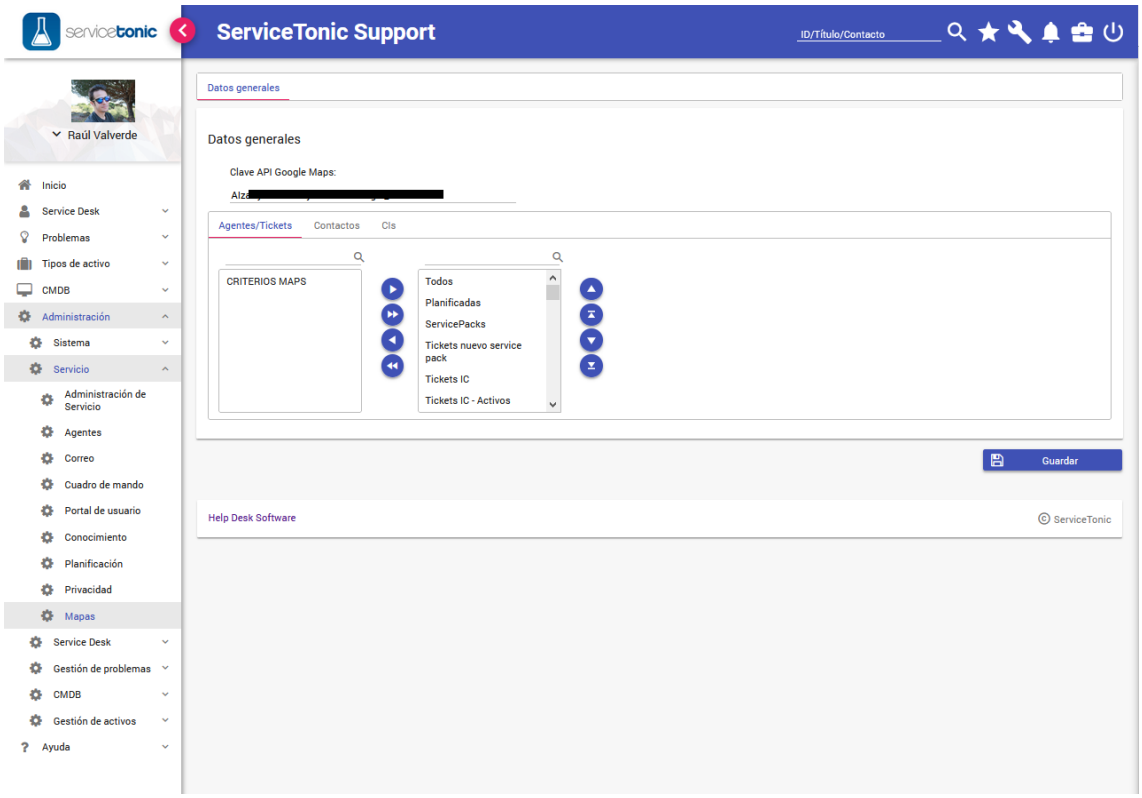


*Figura 2.8.3.2. Página de configuración de las vistas y búsquedas de contactos geolocalizables.*

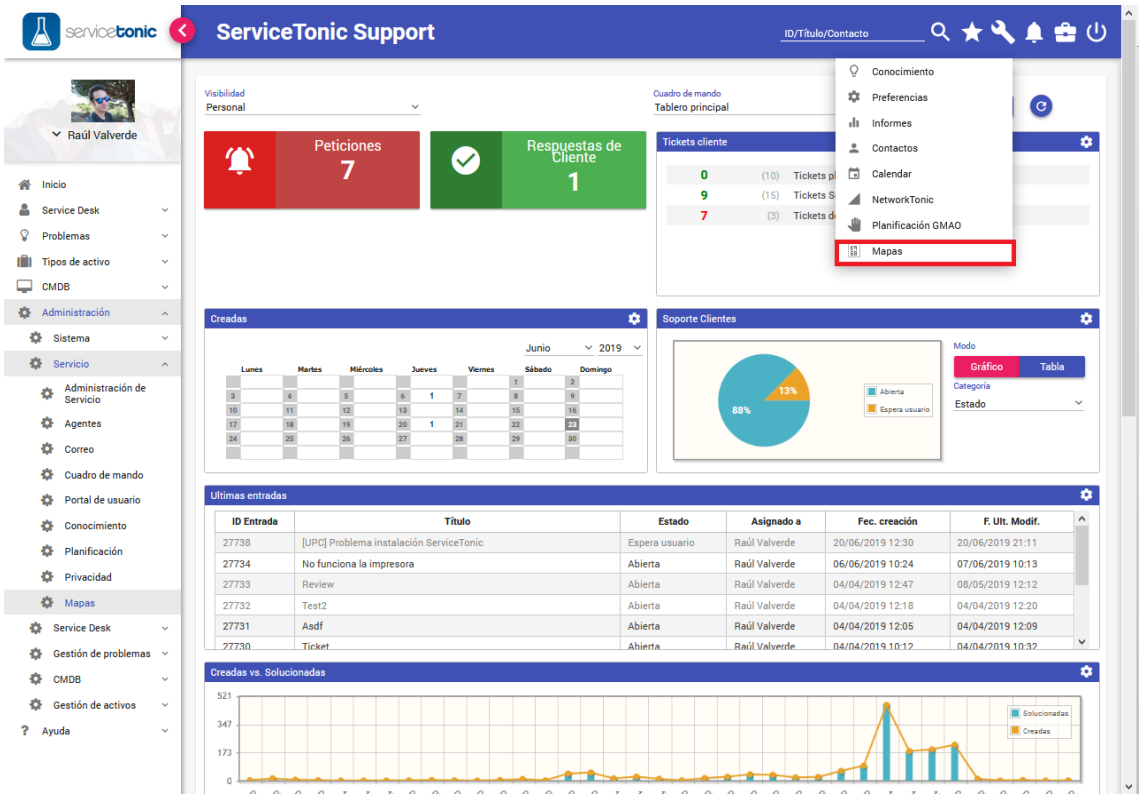


*Figura 2.8.3.3. Seleccionamos la opción de Mapas para entrar en la página de explotación*

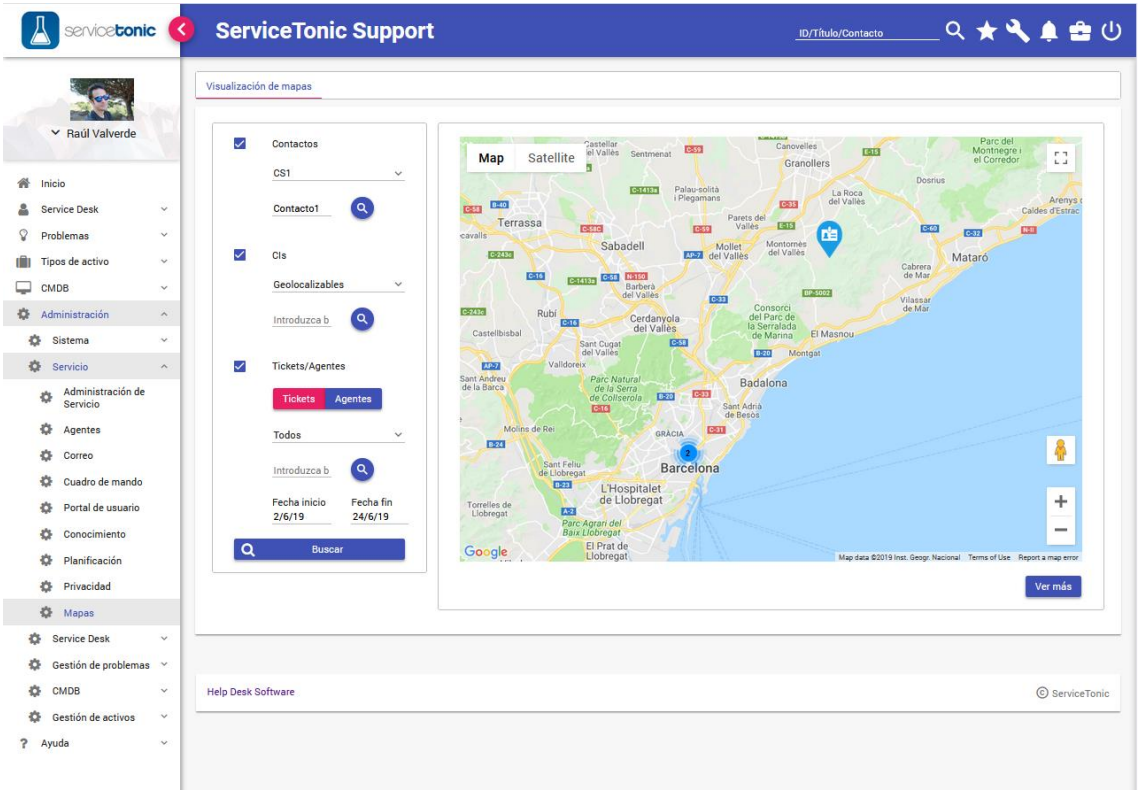


*Figura 2.8.3.4. Mapa de explotación de la información.*

# 3. CONCLUSIONES, PROBLEMAS Y OBSTÁCULOS

## 3.1. PROBLEMAS Y OBSTÁCULOS

Por supuesto, en el proyecto se han producido diferentes problemas y obstáculos que se han tenido que superar o, al menos, bordear.

### 3.1.1. GEOLOCALIZACIÓN EN SEGUNDO PLANO

El primer problema u obstáculo más notorio del proyecto ha sido la cancelación de la mejora *2.4. Geolocalización de agentes automática*. Se perdió bastante tiempo del proyecto intentando buscar alternativas sin demasiado éxito. El motivo de la cancelación de esta mejora vino dada por diferentes factores.

En primer lugar, el motivo más claro es que aun no se ha desarrollado un sistema de geolocalización en segundo plano estable y funcional para aplicaciones web para móviles. Con esto quiero decir, la geolocalización en segundo plano obviamente esta disponible en dispositivos móviles, pero solamente en aplicaciones nativas, es decir, en aplicaciones *Android* o *IOS*.

Las soluciones que se han barajado han sido diferentes *plugins* como Cordova pero éstos eran para frameworks de *JavaScript* como *Ionic* y la empresa no quería desarrollar una app nativa ya que se debería invertir muchísimo tiempo para integrar solamente una funcionalidad como es la geolocalización en segundo plano.

### 3.1.2. PROBLEMAS Y OBSTÁCULOS SECUNDARIOS

A parte del problema comentado en el apartado anterior surgieron una serie de obstáculos y problemas que se tuvieron que ir solucionando a medida que se desarrollaban las diferentes mejoras.

Por un lado, un tipo de problemas bastantes molestos al principio y que con el tiempo supe diferentes formas de solucionarlos fueron los relacionados con el entorno de desarrollo. Como se menciono en el apartado de tecnologías, el IDE utilizado para el desarrollo fue *Eclipse*. Tuvimos un cambio de versión a mitad del proyecto y, debido a esto, hubieron problemas con ambas versiones relacionados con la importación del proyecto y su identificación por parte de *Eclipse* como proyecto *Maven*. A parte de esto, diferentes *bugs* con *Eclipse* impedían que se mostraran los cambios realizados en los ficheros a la hora de hacer la *build* del proyecto.

Por otro lado, tuve que aprender como funcionan los diferentes ajustes y configuraciones del proyecto para integrarlo en el servidor corriendo bajo *Apache Tomcat*, sobre todo a la hora de aplicar el protocolo *HTTPS*.

A la hora de detectar y solucionar los diferentes errores tuve que hacer uso, evidentemente, del *debugger* de *Eclipse*. Comentar que tuve que aprender a utilizar un *debugger* pero en un entorno

real, en un proyecto grande, complejo y con muchas relaciones entre clases. Una vez dominé su uso, la resolución y detección de errores fue muchísimo más rápida respecto al inicio del proyecto.

Relacionado con el desarrollo en sí, un problema bastante complejo y tedioso fue la comprensión del ciclo de vida de JSF y la programación entorno a éste. A pesar de todo, encuentro que JSF es un gran *framework* para la creación de aplicaciones web y acabé adaptándome a su funcionamiento.

Por último, me gustaría comentar sobre el funcionamiento de la metodología de AGILE. Comentar que quizá en el entorno de la empresa ServiceTonic actualmente no sale tan a cuenta invertir tanto tiempo para la implantación de esta metodología. Esto es debido al tamaño del equipo, el cual había semanas que era solamente de cuatro personas. A parte de eso, como en todo proyecto relacionado con la programación el establecimiento de plazos fue bastante laxo debido a las diferentes complicaciones y obstáculos producidos.

## 3.2. CONCLUSIONES GENERALES Y RESULTADOS

En este apartado se sintetizarán las conclusiones y resultados obtenidos a la finalización del proyecto aportando una vista general de lo que ha sido el proyecto, qué puede mejorarse y que puede incluirse.

En primer lugar, cabe remarcar que el propósito de este proyecto era el diseño y desarrollo de un sistema de geolocalización de activos que será integrado en un *software* distribuido para varias empresas a nivel internacional. El sistema debía ser accesible para cualquier plataforma a través de cualquier navegador web por lo que un punto importante era la accesibilidad a este. Por otro lado, había de tenerse en cuenta en todo momento que el sistema iba a ser utilizado por varias personas a la vez y, por lo tanto, tenía que ser lo más rápido posible y, sobre todo, estable.

La solución ha pasado, entonces, por intentar ceñirse a las diferentes peticiones que hicieron los clientes antes del inicio del proyecto. La mejora se ha desarrollado, al igual que todo el resto del *software* de ServiceTonic, siguiendo cada una de las directrices marcadas por el director de proyecto. Esto es bastante importante ya que en el momento de que otro desarrollador quiera continuar mejorando este proyecto pueda hacerlo fácilmente y sin ningún problema. Por lo tanto, implica una serie de factores que se deben cumplir en la medida de lo posible a la hora del desarrollo del código. Para llevarlo a cabo, se han seguido las diferentes convenciones seguidas en la empresa como la estructura del proyecto, la nomenclatura de las diferentes clases y archivos, la estructura del código y, sobre todo, que el código sea legible y que contenga la cantidad adecuada de comentarios para su correcta comprensión.

Otra parte del problema/solución ha sido el sistema de geolocalización externo que íbamos a utilizar. En mi opinión, creo que ha sido la solución más acertada la de escoger los servicios de *Google Maps*, pese a la relativa complejidad de sus tarifas, ya que los servicios que ofrecen y la infraestructura que poseen es muy completa.

### 3.2.1. ANÁLISIS DEL RESULTADO OBTENIDO

Después de haber acabado el proyecto llega la hora de hacer balance sobre los resultados obtenidos.

Para empezar, haría falta remarcar que esta mejora aún no ha salido al mercado, ni si quiera se ha desplegado en la propia empresa, debido a los plazos de tiempo marcados por la durada de este proyecto. Eso sí, el *testeo* de la mejora ha sido bastante elaborado y se comprobado el correcto funcionamiento de cada parte del proyecto constantemente en todo momento.

Debido al hecho de que aún no haya salido al mercado esta mejora, no se ha podido obtener *feedback* de los clientes interesados en esta mejora. La única validación que ha obtenido este proyecto ha sido la del director de este, comprobando el funcionamiento de este y validando todo el trabajo realizado.

En mi opinión, queda bastante trabajo por realizar para mejorar el proyecto, a pesar de que se han cumplido todos los requisitos de las especificaciones sugeridas y dictadas por el director del

proyecto. Entonces, ¿qué se espera después de este proyecto? En primer lugar, en mi opinión, el proyecto está listo para desplegarse en un entorno de producción. Una vez desplegado, escucharía a los clientes que mejoras les gustarían ver en las siguientes versiones y a partir de sus criterios y los propios criterios de la empresa desarrollar las siguientes versiones de esta mejora.

Continuando sobre mi opinión sobre el proyecto, sería necesario mejorar su aspecto visual, ya no sólo de esta mejora sino de todo el producto de ServiceTonic en sí, pero eso es algo que no depende de mí sino de la propia empresa. Por otro lado, toda la parte *backend*, tanto gestión y obtención de datos como eficiencia de esta funcionalidad, creo que es bastante óptima y cumple con las necesidades del abasto del producto de ServiceTonic.

Concluyendo con los resultados obtenidos, creo que se ha realizado un buen trabajo cumpliendo con las expectativas sugeridas por la idea inicial que se tuvo al inicio del proyecto, teniendo siempre en cuenta que el margen de mejora es bastante amplio y que esta mejora es una primera versión.

## 3.3. CONCLUSIONES PERSONALES

Para finalizar, me gustaría expresar mis conclusiones personales que he sacado con la realización de este proyecto.

Soy totalmente consciente de que esta especialidad de la informática, el desarrollo de *software*, no está relacionada con el campo que me gustaría especializarme en un futuro, la administración de sistemas. A pesar de ello, ahora que tengo una edad relativamente temprana me gustaría probar también diferentes campos de la informática para ver y saber en cuál me siento más cómodo.

ServiceTonic me ha acompañado en todo momento en la realización del proyecto, y me ha hecho sentir muy cómodo. He tenido toda la ayuda que he necesitado por parte del director del proyecto aportándome sus consejos y recomendaciones a la hora del desarrollo y la toma de decisiones del proyecto.

Por otro lado, he tenido la oportunidad de aprender una cantidad de conocimientos de desarrollo de *software* y del entorno empresarial enorme y, por ello, me gustaría dar las gracias a ServiceTonic por haberme dado la oportunidad para la realización de este proyecto.

# 4. BIBLIOGRAFÍA

[1] Servicetonic.es (2019) - ServiceTonic | Software de HelpDesk y Gestión de Servicios TI [en línea]:
Disponible en https://www.servicetonic.es/

[2] FIB UPC (2019) - Facultad de informática de Barcelona [en línea]:
Disponible en https://www.fib.upc.edu/

[3] Guia GEP Curs 2018-19 Q2 (2019) - Guia de l'assignatura Gestió de Projectes [en línea]:
Disponible en https://atenea.upc.edu/

[4] Google Maps (2019) - Cambio de precios [en línea]:
Disponible en https://cloud.google.com/maps-platform/user-guide/pricing-changes

[5] Google Maps (2019) - Plataforma de la API de Google Maps [en línea]:
Disponible en https://cloud.google.com/maps-platform/

[10] FIB UPC (2018) - TFG - Informe de Sostenibilitat [PDF]:
Disponible en: https://www.fib.upc.edu/sites/fib/files/documents/estudis/tfg-informe-sostenibilitat-2018.pdf

[11] Oracle – Document Information – The Java EE 5 Tutorial [en línea]:
Disponible en: https://docs.oracle.com/javaee/5/tutorial/doc/docinfo.html

[12] Google Developers – Service Workers [en línea]:
Disponible en: https://developers.google.com/web/fundamentals/primers/service-workers/

[13] HelloTracks – Field Staff Management [en línea]:
Disponible en: https://hellotracks.com/

[14] JustSamIt – IT Asset Management Software & Tracking Tool [en línea]:
Disponible en: https://www.justsamit.com/

[15] FreshDesk – Customer Support Software by Freshworks [en línea]:
Disponible en: https://freshdesk.com

[16] OBS Business School - ¿Qué es Agile y cuáles son los 12 principios de su modelo? [en línea]
Disponible en: https://www.obs-edu.com/es/blog-project-management/metodologias-agiles/que-es-agile-y-cuales-son-los-12-principios-de-su-modelo

[17] JBOSS – Documentación dereferencia de Hibernate [PDF]:
Disponible en: https://docs.jboss.org/hibernate/orm/3.6/reference/es-ES/pdf/hibernate_reference.pdf

[18] Primefaces – Primefaces Showcase [en línea]:
Disponible en: https://www.primefaces.org/showcase/

[19] Primefaces – Documentation [en línea]:
Disponible en: https://www.primefaces.org/documentation/

# 5. ANEXOS

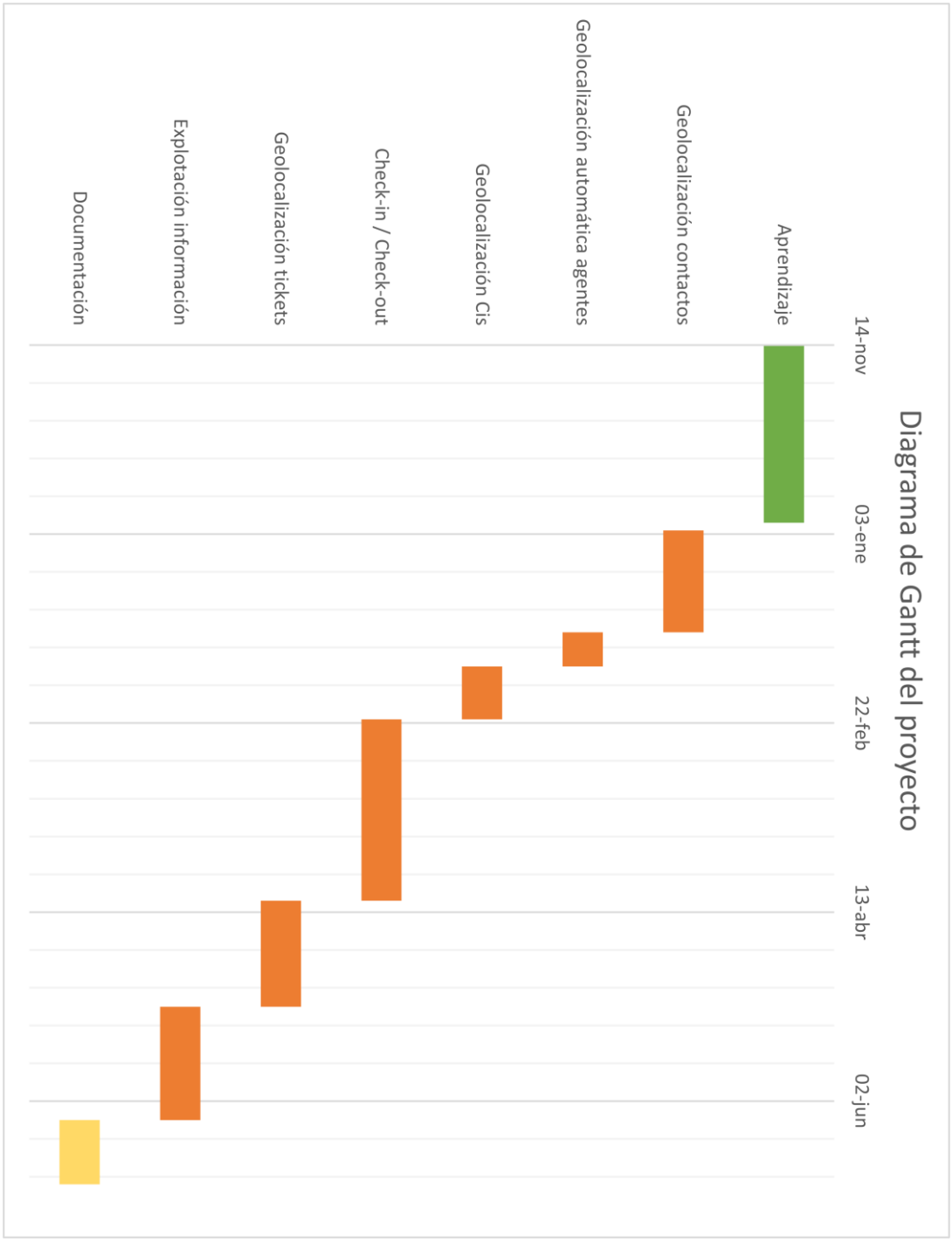


*Figura 5.1. Diagrama Gantt del proyecto*

# 6. NOTAS